Mobile Agent Security and Reliability Issues in Electronic Commerce

CHAN, Hing-wing

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

August 2000


香港中文大學
圖書館系統
28 JUN 2001
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

ABSTRACT of thesis entitled:

Mobile Agent Security and Reliability Issues in Electronic Commerce

Submitted by CHAN, Hing-wing

For the degree of Master of Philosophy in Computer Science and Engineering

At The Chinese University of Hong Kong in August 2000

Mobile software agents are emerging as a major trend of distributed systems in the near future. Different mobile agent frameworks are being actively developed in the research community. Looking forward, electronic commerce and information retrieval are two prospective directions for application of mobile agents. Nevertheless, security and reliability are two crucial concerns for such systems, especially when they are to be used to deal with money transactions. Attacks to agents by malicious hosts are a new and the most challenging part of the problem unsolved, in spite of other security and reliability problems typical of general distributed systems. In this thesis, security and reliability issues of mobile agents, particularly in an electronic environment, are discussed. Models for mobile agent security and reliability have been developed, and a *Shopping Information Agent System (SIAS)* is built as an experimental mobile agent application. Possible security attacks by malicious hosts to agents in the system are discussed, and specific solutions to prevent these attacks are devised. Security of the solutions is analyzed, and the performance overhead introduced is measured. Reliability problems of the system have been identified, and solutions implemented. The reliability improvement gained by the solutions is evaluated according to the reliability model developed in this thesis.

論文摘要

論文題目: 流動軟體代理在電子商業中的安全性與可靠性研究

論文作者: 陳興榮

作為香港中文大學計算機科學及工程學學系
哲學碩士之畢業論文

日期: 2000 年 8 月

流動軟體代理可能是未來分散式系統的主流之一。在世界各地的研究機構，也有不同的流動軟體代理架構在發展中。展望將來，此技術主要的應用將是電子商業及資料提取。然而，安全性與可靠性實為此技術重要之考慮，尤其它要被應用作金錢交易之工具。除了傳統的安全性及可靠性問題外，流動軟體代理在不知名的機器上運作，如何保護它免受有惡意的主機侵襲，是一個最新和最重要的問題。

在本論文中，我們正是要討論流動軟體代理在電子商業中的安全性與可靠性的問題。我們為流動軟體代理發展出新的安全性及可靠性模型。我們亦已實施一個購物資訊代理系統（SIAS），作為一個研究的實驗系統。我們討論 SIAS 個別的安全性及可靠性問題，並針對這些問題，發展出個別的解決方法。最後，我們檢討這些解決方法，並衡量它們的利弊，包括時間上的阻延。

Acknowledgements

In completing the work reported in this thesis, I am most grateful to my thesis advisor, Dr. Michael Lyu, who has been giving continuous support and guidance to me. He has been very resourceful and ready to give a helping hand to me throughout the past two years.

Besides, my thesis examiners, namely Prof. Kam-wing Ng and Dr. Man-hong Wong of the Chinese University of Hong Kong, and Dr. Francis Lau of the Hong Kong University, have been very generous to give their valuable insights on the subject, and comments for refinements of this thesis.

Many thanks go to Tsz-Yeung Wong and Caris Ka-Ming Wong for helping the work of implementing SIAS and taking the measurements for the security and reliability experiments, which is a very significant part of this thesis.

Finally, I am also obliged to my colleagues, namely Wing-Kai Lam, Steve Ka-Lung Chong, Sam Ka-Po Ma, Chris Ting-On Lee, Kwong-Wai Chen, Wai-Chiu Wong, Wai-Ching Wong, Po-Shan Kam, Yin-Hung Kuo, and Clara Tsui-ying Law, in the Department of Computer Science and Engineering. Without them, the life of my Master studies would not be that fun.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Mobile Agents and the Problems

Mobile agents are autonomous software agents that travel in a computer network to execute and perform tasks on different hosts for their owners. It can be classified as a branch of mobile code, including also remote evaluation, code-on-demand, which is an emerging trend for distributed systems.

Autonomous mobile agents not only benefit users by allowing delegation of tasks, like what other autonomous agents do, but also bring practical advantages such as reduced network communication costs for distributed tasks. A lot of mobile agent platforms have been developed around the world, such as Aglets [IBM99] from IBM, Concordia [Mit99a] from Mitsubishi, and the Mole [IPV98] from University of Stuttgart. Prospective applications of mobile agents include electronic commerce, information retrieval and network management.

Nevertheless, security is one of the blocking factors of the development of these systems. The problem of mobile agent security can be divided into two parts. One of them is the protection of hosts against agents. This is similar to protecting hosts from being attacked

by incoming Java applets, which may be malicious. The other one is the protection of agents against hosts. This is the main unsolved security problem for mobile agents, because the possible existence of malicious hosts that can manipulate the execution and data of agents [Hoh98b] and the lack of a trustworthy computing base [ST98] add new complexities to the problem.

On the other hand, despite the growing efforts being invested on the development of mobile agent systems, these systems have not proved to be reliable. Problems such as host failure, communication failure, and loss of agent and/or their states exist as they similarly do in other distributed systems. Solutions such as replication, consensus protocols, agent rerouting, and agent persistence are yet to develop, though there are already some work done [WPW98].

As an alternative for distributed system development, mobile agent systems must be sufficiently secure and reliable; otherwise it is unlikely that application developers would switch to mobile agents from the existing, more secure and reliable, technologies, for example, client-servers.

This thesis examines both security and reliability issues for mobile agents, particularly in an electronic commerce environment. It is obvious that in such an environment, security and reliability of mobile agents become extra-important, because money transaction is involved.

Although both security and reliability are important, the primary interest of the work reported in this thesis, when it first began in 1998, was on security only. The work on reliability reported in this thesis did not start until early 2000, when it was found that the mobile agent system described in this thesis did not prove to be reliable. Only since then we realized that there should be more work on mobile agent reliability. Therefore, the weights of security and reliability issues in this thesis are not equal, with security being heavier.

1.2 Approach

In general, security and reliability issues can be addressed in two ways: one way is theoretically and generically; the other is implementation-wise and application-specifically. This thesis takes a dual approach.

Firstly, effort is spent on exploring and developing new models for protection of general mobile agents against malicious hosts and system failures. This is motivated by the lack of theoretical models for mobile agent behavior, both in terms of security and reliability, which should be necessary for further discussion of the problem. This direction is supposed to give a generic solution, or at least a starting point of such, to the problem, but may lack application-specificity.

Secondly, to be more focused, a specific application (in electronic commerce) is chosen. Mobile agents are implemented for such application. The security and reliability issues of such implementation are studied and enhancements are added to the specific implementation. This direction is supposed to complement the first one by uncovering more practical issues of the problem.

An important point to note, however, is that the two directions are intended to be independent, so that more issues can be uncovered. Therefore, it should not be expected that the results obtained from the theoretical direction necessarily correlates with the results obtained form the implementation direction, though it actually does for the reliability modeling.

1.3 Contributions

This thesis makes three contributions:

- It gives a survey on the security and reliability issues of mobile agent systems;
- It defines a new model for evaluating the security of mobile agent systems quantitatively; and

- It develops the Shopping Information Agent System (SIAS), based on the Concordia mobile agent platform, on which some security and reliability enhancements have been designed and implemented.

Although mobile agent technology has been developing for more than a decade, and is gaining increasing attention from the general research community, its security and reliability issues are still new, especially on the security challenge to protect mobile agents against malicious hosts, to many researchers. Therefore, it justifies to survey on these issues before digging into the problems.

After attaining some knowledge about the problems, solutions to the problems are sought. In observing the analogy between security and reliability, a novel, quantitative, model for evaluating security of mobile agent systems is developed. This model may be used for deciding how trustworthy a mobile agent system can be, given the time an agent spends on each host, possibly malicious. Conversely, it may be used to decide how long an agent should stay on a host such that it can be safe.

Finally, to experiment with mobile agent technology, a system named *Shopping Information Agent System (SIAS)* is built using the Concordia architecture. This is a pilot system of mobile agent application in electronic commerce. The system is useful to collect and compare the prices of a set of products specified by users from different seller hosts in an electronic market. The ultimate goal of building SIAS is to push it as real an application as possible, therefore security and reliability issues of the system must be addressed. Possible attacks by malicious to the system, and solutions to protect the system against these attacks are devised and implemented. Possible failures of the system are also discussed, and some fault-tolerance measures are implemented.

1.4 Organization of This Thesis

The thesis is organized in the following way:

- Chapter 1 (this chapter) is an introduction of the thesis. It gives a brief description of mobile agent technology and the security and reliability problems, and outlines the contributions and organization of this thesis.
- Chapter 2 gives an overview of the evolution of the mobile code paradigm. Mobile agents are really one kind of mobile code. To understand mobile agents and their values, it is worth knowing different mobile codes, namely remote evaluation, code-on-demand, and mobile agents, as a whole, and making a comparison between them and the already-in-use client/server paradigm.
- Chapter 3 gives a more focused discussion of the security and reliability issues of mobile agent systems. It states some security and reliability problems specific to mobile agents, and reviews the current solutions to these problems. This discussion has first been presented in [CL00].
- Chapter 4 introduces a security model of mobile agents against malicious hosts based on reliability theory. The model is developed by applying reliability modeling technique to mobile agent systems, assuming the model of attack described in [Hoh98b]. This idea first originally published in [CL99]. Furthermore, a simple reliability model for mobile agent system is also derived.
- Chapter 5 describes the Concordia mobile agent platform. This deserves a chapter of discussion because in the next chapter, a mobile agent application will be developed based on this platform. The materials mainly come from [Mit99a], they are restated here to make this thesis more complete.
- Chapter 6 gives an overview of SIAS, the mobile agent application implemented for security and reliability experiments. The security and reliability problems and solutions of SIAS are addressed, and then an evaluation of the respective solutions for SIAS is presented.
- Finally, Chapter 7 concludes the paper and suggests some directions for future work.

Chapter 2

The Mobile Code Paradigm

Mobile agents can be classified as one category of the broader class of *mobile codes*, which is an aggregation of the *remote evaluation*, *code on demand*, and *mobile agent* paradigms. The mobile code paradigm is an emerging programming paradigm that complements the conventional client/server paradigm, by reducing network usage. To understand mobile agents better, it is a good idea to know how the mobile code paradigm evolves. This chapter describes a classification of these three types of mobile codes, which are easily confused with each other, points out the additional sophistication of mobile agents compared with others, and describes some applications and implementations of the paradigms.

2.1 Mobile Code: an Alternative to Client/Servers

Distributed applications are applications that involve the coordination of two or more computers, geographically apart and connected by a physical network. Most distributed applications we see today, such as email, network news, file transfer, and Web browsing, are deploying the *client/server paradigm.* In the client/server paradigm, an application is divided into two processes, a client process running locally that asks for services and a server process

on a remote site that gives services to the client. The client and server processes must communicate with each other in order to carry out their tasks successfully. Communication is done by means of message exchange. There are at least two problems with the client/server paradigm:

- It has a high network bandwidth requirement due to the large number of messages exchanged.
- It usually requires users to response to computation results interactively, under different situations. Neither the client nor server would make decisions for users autonomously.

In view of the deficiencies of the client/server paradigm, *the mobile code paradigm* has been developed as an alternative approach for distributed application design. In the client/server paradigm, programs are geographically stationary, meaning that they do not move to another machine, and only run on the machines they reside on. (Therefore, the two processes must communicate continuously.) The mobile code paradigm, on the other hand, allows programs to be transferred among and executed on different computers. By allowing code to move between hosts, programs can interact on the same computer instead of over the network. Therefore, communication cost can be reduced.

Besides, the mobile programs can be designed to work on behalf of users autonomously. In this case, the mobile programs are called *mobile agents* [Whi98]. The autonomy of mobile agents allows the users to delegate their tasks to the mobile agents, and not to stay continuously in front of the computer terminal.

The promises of the mobile code paradigm bring about active research in its realization. Supporting mobile code technology is also fast developing. Experimental mobile code systems that allow programs to move on a computer network have become available for download on the Internet [AS98]. However, these systems are not yet popular. Neither is the mobile code paradigm commonly adopted. Most researchers agree that the hurdle is security concerns [GBH98]. To gain general acceptance of the mobile code paradigm and systems,

especially from *application developers*, there must be strong evidences for a certain security level for applications developed using such paradigm and systems. As this chapter proceeds, we will see some of these security guarantees are becoming evident, while some others not.

2.1.1 Classification of Mobile Codes

The mobile code paradigm is actually a collective term, applicable wherever there is mobility of code. There are different classes of code mobility. In essence, Ghezzi and Vigna identified three of them, namely *remote evaluation*, *code on demand* and *mobile agent* [GV97]. The classification, together with the client/server paradigm, is summarized in Table 2.1.

Paradigm		Local side	Remote side	Computation takes place at
Client/server		-	Know-how	Remote side
			Processor	
			Resources	
Mobile code	*Remote evaluation*	Know-how →		Remote side
			Processor	
			Resources	
	Code on demand	←	Know-how	Local side
		Processor		
		Resources		
	Mobile agent	Know-how →		Remote side
		Processor →		
			Resources	

Table 2.1. Ghezzi and Vigna's classification of mobile code paradigms.

In particular, the *know-how* in the figure represents the code that is to be executed to accomplish the specific task. In the mobile code paradigms (remote evaluation, code on demand, and mobile agent), the know-how moves from one side to another side, where the

computation takes place; while in the client/server paradigm, the know-how is geographically stationary on the remote (server) side. *Resources* are the input and output for the code; *processor* is the abstract machine that carries out and holds the state of the computation. The arrows represent the directions in which the specific item should move before the required task is carried out.

Remote evaluation has the know-how on the local side, but the resources and processor on the remote side. The know-how is to be moved from the local side to the remote side for computation to carry out. The opposite happens for code on demand where the resources and processor reside on the local side, but not the know-how.

The mobile agent paradigm adds complexity to the remote evaluation paradigm by allowing also the processor to move around with the know-how. More clearly speaking, the code moves from the local side to the remote side together with the processor that holds the state of execution of the code. For a single-hop application (in which the code has only one destination), this could be in effect equivalent to remote evaluation, because we can always modify the initial state of the code to that of the mobile agent. However, the merit of the mobile agent approach over the remote evaluation is that it allows multi-hop applications (in which the code has more than one destinations) to be developed. The know-how carries its own processor (and thus the state of execution) throughout the journey, and therefore can make different execution decisions according to different execution results on previous destinations. This idea is actually an extension of the intelligent agent concept applied in a distributed execution environment [RN95]. Table 2.2 visualizes the behavior of different mobile code paradigms in a two-hop application. Notice that, however, there are seldom multi-hop applications for remote evaluation and code on demand.

Ghezzi and Vigna's classification is found to be comprehensive and representative of most existing mobile code paradigms, and we will base the following discussion on this classification.

Paradigm		Local side	Remote side 1	Remote side 2
Mobile code	*Remote evaluation*	Know-how →	→	→
			Processor	Processor
			Resources	Resources
	Code on demand	←	←	Know-how
		Processor	Processor	
		Resources	Resources	
	Mobile agent	Know-how →	→	→
		Processor →	→	→
			Resources	Resources

Table 2.2. Mobile code paradigms in a two-hop application.

2.1.2 Applications of Mobile Code Paradigms

While the mobile code paradigm is not common among application developers, it is already implicitly used in quite many common applications. For example, when you use the Unix `rsh` utility to, say, remove (`rm`) a file in a file system mounted on the remote host, you are actually sending a code (the "`rm`" command) from the local side to the remote side. Meanwhile, the processor (the remote-side command interpreter, or Unix shell) and the resources (the file) resides on the remote side. This is in effect equivalent to remote evaluation.

The Java programming language has provided a big step for the code on demand paradigm. When a Web browser downloads a Java applet (code) from a remote host, and run the applet on the local Java virtual machine (processor), with for example, local file inputs (resources), he/she is actually running code on demand.

The more complex mobile agent paradigm, however, is less commonly applied in distributed applications. In spite of this, the enlightening idea of code autonomy in a de-centralized system has directed researchers to increasing efforts of making the paradigm realistic. [Whi98] suggested many mobile agent applications in everyday life. One pro-

classical example in electronic commerce is autonomous bargaining and shopping. Besides, the mobile agent concept is being used heavily in telecommunication architectures [PK98]. It is also possible to use mobile agents for network management [BGP97], QoS implementation [OOC97], and dynamic distributed service trading [BP98].

No matter which category of mobile code paradigms is employed in application, users are always specifically reminded of the underneath security risks. The `.rhosts` is commonly regarded as a security hazard in many systems, and some system administrators simply ban the use of the `rsh` facility; browsers pop up warning messages when a Java applet is downloaded from an anonymous site; and agent tampering is being the hindering stone of exercising mobile agents in the world of electronic commerce. In the coming sections, we will have a closer examination of security concerns for the mobile code paradigms.

2.1.3 Supporting Implementation Technologies

A paradigm (both mobile code and client/server) would be merely a matter of talk if there were no implementation technology that supports its realization. Fortunately, active research has been taking place to realize the client/server paradigm since the past few decades, and these few years for the mobile code paradigm [AS98].

Together with the classification of mobile code paradigms discussed in Section 2.1.1, Ghezzi and Vigna also gave a classification of the supporting implementation technologies into message-based, weakly mobile and strongly mobile technologies [GV97]. Example of message-based technology are Remote Procedure Calls (RPC); weakly mobile technology the `rsh` facility, Obliq and Mole; and strongly mobile technology Telescript and Agent Tcl (which is now called the D'Agents). However, we feel this classification inadequate because it does not cover the already-popular object-based technology, namely the Common Object Request Broker Architecture (CORBA). Moreover, there seems no

sharp cut-off between weakly mobile technology and strongly mobile technology. Therefore, we slightly modify the classification as follows:

- *Message based* technology: technologies that allow an executing unit to send messages to and receive messages from a remote executing unit. Again an example is the RPC.
- *Object based* technology: technologies that allow an executing unit to communicate with another executing unit on a remote side in the form of object invocation. Examples are implementations of the CORBA specification, such as Orbix [Bak97] and Visigenic [Inp98].
- *Mobile* technology: technologies that allow an executing unit to move their code to a remote side, possibly carrying their execution state when they move. This can be taken as the union of weakly mobile technology and strongly mobile technology described in [GV97]. Examples are the Aglets, Mole [IPV98], and Odyssey [GM98].

Ghezzi and Vigna argued in [GV97] that message-based technology and mobile technology are well suited for client/server architecture and mobile code architecture respectively. We claim that object-based technology is at present better suited for client/server architecture. This is because they do not provide services for transferring code among different locations, up to our knowledge. However, the OMG is drafting a specification of Mobile Agent System Interoperability Facility (MASIF) for their CORBA architecture [MBB+98]. This points some light on future of CORBA support for mobile agent implementations. Nevertheless, we assume message-based and object-based technologies are used for client/server application development, while mobile technology for mobile code development. Notice that this classification, however, by no means cover all existing technologies for development of distributed applications.

Different implementation technologies come with different sets of security features, which help application developers to satisfy certain security requirements of the application being developed. In a later section, we will examine and compare these features.

2.2 The Problems of Mobile Code

Most researchers agree that the bottleneck for mobile code deployment lies on security. Although reliability should also be an important concern of mobile agents, it does not bring as much new challenges as security does. In this subsection, we focus on the security problems of mobile code. Section 2.2.1 is a background for general security issues in distributed systems. These issues are extended to the mobile code paradigm in Section 2.2.2, and the supporting implementation technology in Section 2.2.3

2.2.1 Security Issues in Distributed Systems

Before we go to discuss the security concerns for mobile code paradigms, and security features of mobile code technologies, we first state our scope on what we mean by security in the later sections. In general, there are four requirements for security in an information system: *availability*, *confidentiality*, *integrity*, and *authenticity* of both code and data. It is common to consider these security requirements in three aspects [Sta99]:

- *Security attack*: an action that compromises the security requirement of information owned by an organization. Examples of security attacks are interruption, interception, modification, and fabrication;
- *Security mechanism*: a mechanism that is designed to detect, prevent or recover from a security attack. One of the major mechanisms, among others, is the use of cryptographic techniques. A very good treatment on cryptography applied to information security is given in [Sch96].
- *Security service*: a service that enhances the security of the data processing systems and the information transfers of an organization. The service is intended to counter some security attacks, and it makes use of one or more security mechanisms to provide the service.

We identify, in the typical process of application development, there are two central concerns before the mobile code paradigm can be deployed and mobile code technologies utilized in application development:

MCS1: An application developed using the mobile code paradigm can be as secure as the same application developed using the conventional client/server paradigm.

MCS2: With the same security requirements, it is as easy to implement an application with the mobile code paradigm as to implement it with the client/server paradigm.

The identification of these concerns is natural and straightforward. A typical application development process consists of the *requirement analysis*, *design*, *implementation*, and *testing* phases [Som92]. Given a set of security requirements in the requirement analysis phase, MCS1 concerns with the acceptance of the mobile code paradigm in the design phase of the application development process, while MCS2 concerns with the acceptance of the supporting technology in the implementation phase of the process. The testing phase is, however, less relevant to the acceptance of mobile code.

Restating in the usual security terms, and comparing with the client/server paradigm, MCS1 is equivalent to saying that the mobile code paradigm must not bring, to an application being developed, additional security attacks that have no corresponding security mechanisms to counter with. On the other hand, MCS2 is equivalent to saying that there must be easy-to-use security services that can be employed to implement security mechanism to meet certain security requirements of an application.

In Section 2.2.2, we explore possible security attacks to applications developed with the mobile code paradigm, and present an review on the development of the underlying security mechanisms that would be able to counter these attacks. Then we make a comparison with the client/server paradigm to see if one paradigm is more vulnerable to security attacks or not. This would lead us to the current status of the validity of MCS1. In

Section 2.2.3, we survey the security services provided by different supporting technologies, and make a comparison to see whether adequate security services are provided to application developers by these technologies. This leads us to a review on the current status of validity of MCS2.

2.2.2 Security Concerns of Mobile Code Paradigms

In this section, our aim is to discuss some possible security attacks (Section 2.2.2.1) to different mobile code paradigms, and possible mechanisms (Section 2.2.2.2) against these attacks. To address MCS1, we compare these attacks and mechanisms with the attacks and mechanisms corresponding to the client/server paradigm in Section 2.2.2.3.

2.2.2.1 Security Attacks

As discussed in Section 2.2.1, a security attack is an action that compromises the security requirements of an application. Applications developed using different paradigms are subject to different attacks, which would be discussed in the following paragraphs.

I. Security Attacks to Client/Server Paradigm

Conventionally, the client/server paradigm assumes the security model of an "information fortress". In the information fortress model, the local computer is assumed to be a secure premise for code and data. Only the communication channels and remote sites would be suspected to be malicious. This effectively limits the source of security attacks to outsiders of the local machine, and eliminates the following two types of attacks:

- attacks to the client on the client side
- attacks to the server on the server side

Therefore, the main, among other, possible attacks are:

- pretending the server to the client

- pretending the client to the server
- eavesdropping on the communication channel
- forging messages to the client or server

And therefore, the main challenges are to:

- establish the appropriate trust relationship between the client and the server
- securing the communication channel from being eavesdropped

Over the past few decades of time, since the time when the client/server paradigm was first introduced, efforts have been spent on developing mechanisms to defense these attacks, with rewards. We are going to outline some significant mechanisms of such in section 2.2.2.2.

II. Security Attacks to Mobile Code Paradigms

While the security fortress model is usually assumed in the client/server paradigm, it also applies to the remote evaluation and code on demand approaches, with some additional considerations.

- *Remote evaluation*

In the remote evaluation paradigm, the security fortress model may also apply, with the addition of the following three concerns:

i. the local side must make sure it is sending the code to the correct site

ii. the remote side must make sure it is receiving the code from the correct site

iii. the remote side must make sure the code is not harmful to run

Therefore, apart from building trust between the sender and the receiver, there is an additional challenge of verifying the code to the defending mechanism. Again, we will see in section 2.2.2.2 that this is quite well developed.

- *Code on demand*

Code on demand is similar to remote evaluation, as we see from Figure 2.1. The only difference between these two approaches is the swapping around of the roles of the remote side and the local side. Therefore, the challenge is also on code verification.

- *Mobile agents*

Mobile agent is the most challenging area of mobile code security, due to the autonomy of agents. Mobile agent security is usually divided into two aspects: host security and agent security. Host security deals with the protection of hosts against malicious agents or other hosts; while agent security deals with the protection of agents against malicious hosts or other agents.

For host security, the security fortress model can still apply, however agents may co-operate together to perform a complex attack [GBH98]. For agent security, on the other hand, there is a lack of trusted hardware for agents to anchor security with [Tsc99]. There are two branches of possible attacks to agents:

i. *data tampering*: a host or another agent may modify the data or execution state being carried by an agent for malicious purpose

ii. *execution tampering*: a host may change the code executed by an agent, or rearrange the code execution sequence for malicious purpose

We will examine more details of mobile agent security in Chapter 3.

2.2.2.2 Security Mechanisms

To defense security attacks, we have security mechanisms. This subsection reviews the common security mechanisms that can be used to protect a system from attacks mentioned in Section 2.2.2.1.

I. Security Mechanisms against Attacks to Client/Server Paradigm

We see from Section 2.2.2.1 that the main security challenges of the client/server paradigm are the mutual trust building between clients and servers, plus the protection of messages in transit. These problems can be satisfactorily solved by cryptographic techniques:

- *Authentication protocol*, such as Kerberos [MIT99b]: the client and server processes are first registered to a trusted third-party authentication server (AS). When the client needs to open a connection with the server, it asks to AS to issue a "ticket", which can be used to prove its identity to the server. While Kerberos is an arbitrated approach [Sch96], SSLv3 [FKK96] is a self-enforcing alternative.
- *Encryption of messages* in transit: the client and the server, after mutual authentication, can be assigned a per-session key, and encrypt all messages exchanged so that the messages would not be understood even if they are being caught by an eavesdropper.

Actually, these mechanisms (authentication, encryption) are already extensively employed in existing client/server applications. A lot of details can be found in [Sch96] and [Sta99].

II. Security Mechanisms against Attacks to Mobile Code Paradigms

In parallel with the increased possible attacks to mobile code paradigms, more mechanisms are required to secure mobile code applications. We see from section 2.2.2.1 that the main additional challenge to security of remote evaluation, code on demand, and also host security of mobile agents, is the verification of the received code. Some significant approaches to this problem are the *sandbox model* and *verification techniques*.

In the sandbox model, the code or agent received from a remote side can only access a dedicated portion of system resources. Access to resources other than the dedicated ones are not allowed to unauthorized principals. Therefore, even if the received code or agent is malicious, and perform some attacks successfully, the damage is confined to the resources

dedicated to that code or agent, and would not harm the operation of others. This is the main security mechanism of the Java programming language, and Java-based systems such as Aglets [KLO97].

In addition to sandboxes, received code or agents are filtered through a code verifier, which checks the integrity of code, for example, no access of out-of-bound memory and type safety. A host can also limit the period of time for a specific received program to run, and thereby reducing the risk of time-consuming attacks. Besides, code may be digitally signed for hosts to verify that they are actually from the intended senders.

Sandboxes and verification techniques serve to satisfy most of the host security requirements of mobile code applications. These techniques are well known enough and generally accepted. On the other hand, for agent security, there is not a very well established mechanism to protect an agent from being tampered with. However, some approaches have been proposed, and they can be classified into two categories:

- *Agent tampering detection*: techniques that aim at detecting whether an agent's execution or data have been tampered with along the journey. Some possible approaches are range verification, addition of dummy data items and code, and cryptographic watermarks.
- *Agent tampering prevention*: techniques that aim at preventing agent code or data being tampered with. Two possible approaches are execution of encrypted functions [ST98] and time-limited black-boxes [Hoh98a].

We will go into more details of mobile agent protection in Chapter 3. One point to note is that, although these approaches open new areas in computer security, and have caught quite much attention from the research community, none of them is complete in agent protection yet. The stage of agent protection is in its infancy, compared with the maturity of protection for hosts and client/servers.

2.2.2.3 A Security Comparison between Paradigms

From section 2.2.2.1, we see that the mobile code paradigm is actually facing more security attacks than the client/server paradigm; and the mobile agent approach faces more possible attacks than do remote evaluation and code on demand. From section 2.2.2.2, we see that mechanisms are being developed to parallel the attacks. However, combining the two sections, we see the trend in Figure 2.1.

Attacks to the client/server paradigm are least possible, due to the assumption of the security fortress model. Mechanisms are well established to defense these attacks. Remote evaluation and code on demand are subject to more attacks, but the mechanisms against these attacks are also quite well established. This is consistent with the popularity of Java applets, which is a typical example of code on demand. Mobile agents, on the other hand, are facing more attacks, yet the defending mechanism is not very well established. Therefore, from the view of an application developer, remote evaluation and code on demand are acceptable for software design; however, the acceptability of the mobile agent approach is still in question, i.e., MCS1 is currently invalid.

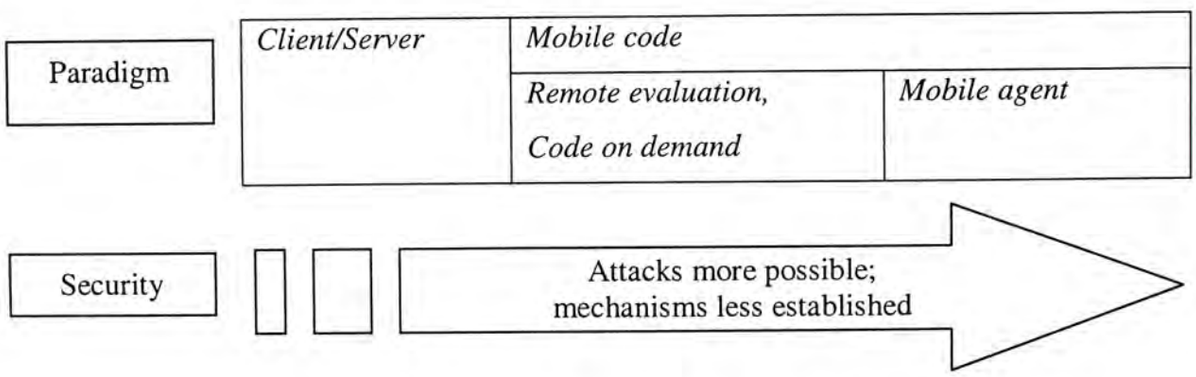


Figure 2.1. Security trend of mobile code paradigms.

2.2.3 Security Features of Implementation Technologies

Besides MCS1, MCS2 is also an important proposition to be valid for the acceptance of mobile code. That is, with the same security requirements, it is as easy to implement an

application with the mobile code paradigm as to implement it with the client/server paradigm. In this section, in order to evaluate the current status of this proposition, we examine the security services of different implementation technologies described in Section 2.1.3.

2.2.3.1 Security Services of Message-based Technology

The most typical example of message-based technology is the Sun RPC. Recent versions of Sun RPC come with the secure RPC services, which provide authentication functions to application developers with four options (no authentication, system authentication, DES authentication and Kerberos authentication). Library functions that implement the SSLv3 are available in both the commercial and public domains. Effort is made to standardize the application program interface of security services is the proposed standard of a Generic Security Service Application Program Interface v.2 [Lin97].

One point to note about the security services of message-based technology is that they usually exist as some kind of add-ons to applications and systems. The availability of such services is all up to the preference of system administrators, and the use of such services is optional to the application developers. Security services are not embedded to the system.

2.2.3.2 Security Services of Object-based Technology

A typical example of object-based technology is the CORBA architecture. Security services are provided in CORBA as a CORBA service. The CORBA Security Service Specification [OMG98] requires implementation of objects such as Credentials, Principal Authenticator, Security Context, and Access Control. These objects support implementation of authentication, authorization and security auditing at a higher object level than the "procedure level" of Kerberos and SSL.

However, to our knowledge, there does not exist such implementation yet. Some vendors simply add their own security add-on for their ORB product (for example, the SSL pack for Visibroker). Nevertheless, it is expected all implementation of CORBA should eventually follow the specification. This means that the security services would be more embedded into the system, and the use of such services can be at a higher level than how it is being used in message-based technology.

2.2.3.3 Security Services of Mobile Technology

Since mobile technology is still in early research and development stage, we do not have much information about the security services of these systems. Some of the little is that Aglets and Odyssey are employing the Java security model (sandboxes and signed applets) for host protection. There is no implementation of agent protection, though [KLO97].

Nevertheless, it is expected that the security services of mobile technology would also be embedded into the system, and available at a high level of abstraction.

2.2.3.4 A Comparison of Technologies on Security Services

Through sections 2.2.3.1 to 2.2.3.3, we examined the main features of different technologies that are related to meet security requirements. Figure 2.2 summarizes the trend of security services in different technologies we find.

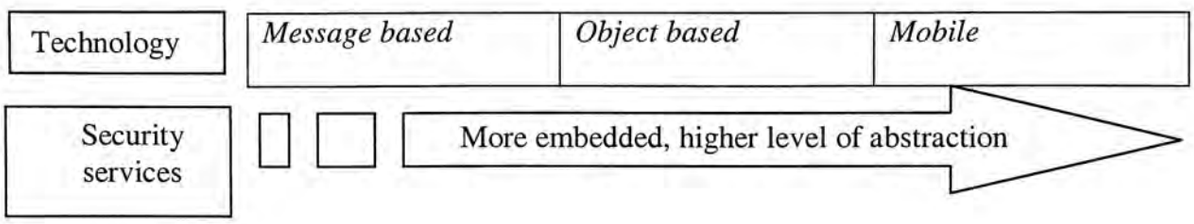


Figure 2.2. Security services of different implementation technologies.

The rising level of abstraction from message based technology, object based technology to mobile technology indicates that security services are increasingly aimed at "user-friendliness" to application developers. This validates part of MCS2. Since we do not have much information about the details of such services, and maybe there is not yet actual implementation of such services, we cannot justify the remaining part. However, if the security mechanisms described in Section 2.2.2.2 can really be implemented, and available at a high level, hopefully MCS2 would be valid, and mobile technology could then be accepted by application developers.

2.3 Chapter Summary

Throughout this chapter, we described mobile agents as a branch of mobile code. We discussed different mobile code paradigms, identified their key differences and the respective supporting implementation technologies. We studied the major problem, security, of mobile code, and figured out that mobile agent security and the supporting mobile technology is the least developed. This justifies the value of our study on mobile agents.

In the next chapters, we will focus on mobile agents only. We will study the security and reliability issues of mobile agents, and apply mobile agents to an electronic commerce environment.

Chapter 3

Mobile Agents, Its Security and Reliability Issues

In contrast to the previous chapter, this chapter is devoted to discuss mobile agents only. To further recognize the value of mobile agents, we first discuss the advantages and applications of mobile agents. Then, we will discuss in more details the security and reliability problems of mobile agents.

3.1 Advantages and Applications of Mobile Agents

Let us consider the following scenario: a user is looking for a particular piece of information, from different hosts in a network. For instance, the user may want to search for the prices of a particular product at different web sites in an electronic commerce network. The user may use the conventional client/server technology to query different web sites interactively by his own, each time connecting to a different web server. During the time, the user needs to stay in front of the computer terminal to disconnect from one server and connect to another. On the other hand, he may send out a mobile agent, which would travel around the different servers, and retrieve the information for the user from the different hosts. During the time,

the user may disconnect his local computers and go away, before he returns to the computer terminal to see the agent report to him/her.

The two choices are illustrated in Figures 3.1 below.

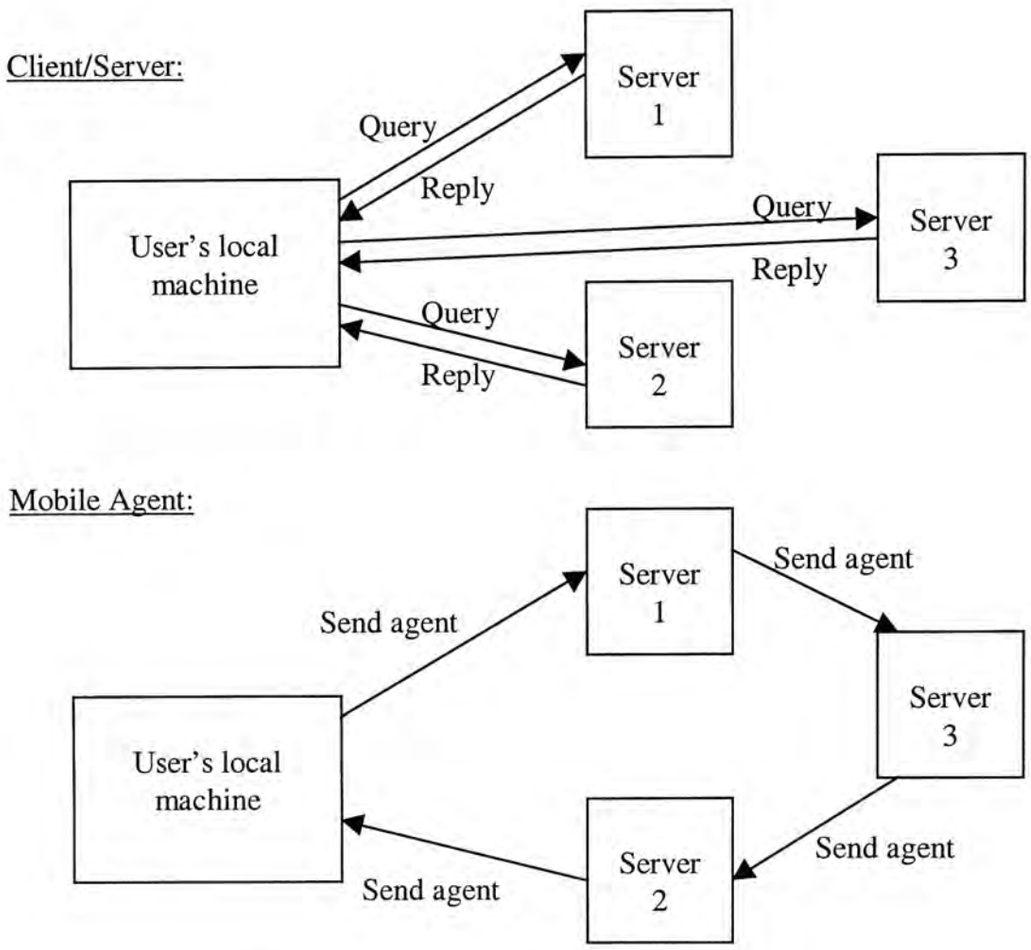


Figure 3.1. Mobile agent against client/server.

If each arrow in the figure represents one message sent, we can see that using a mobile agent actually saves two messages in a network of three servers, compared with using client/server. In general, if there are *n* servers, *(n-1)* messages can be saved using mobile agents.

Actually, there are several good reasons for mobile agents [LO99] against the conventional client/server paradigms, for example:

- *reduced network usage*: with code mobility, we can send a program to a remote host to execute, and get back the result after the program execution is finished. Therefore, there

are only two message transfers (sending the program and receiving the result), and any intermediate communications as in the client/server paradigm can be done locally on the remote site, therefore network usage is reduced;

- *load sharing*: an agent can adapt dynamically. It can find a host that has a lower workload and execute there, thus workload sharing among hosts is achieved; and
- *delegation of time-consuming tasks*: with autonomous mobile agents, users can send programs that execute on and travels autonomously among different hosts in a computer network. Therefore, users can delegate time-consuming tasks, such as collecting information from different hosts, to these programs

These advantages make mobile agent a good paradigm for applications where network connection is slow, for example, in a mobile computing environment; or when the task is very time-consuming, for example, when retrieving information from a lot of different hosts, which respond slowly.

One prospective use of mobile agent is in electronic commerce. One can imagine a mobile agent is sent by a user to shop around different hosts in an electronic market, and perform the best transaction possible for the user. However, this will not be feasible before the security and reliability problems of mobile agents are solved. The remaining of this chapter discusses these two problems.

3.2 Security Concerns of Mobile Agents

As discussed in Chapter 2, any distributed system is subject to security threats such as eavesdropping, corruption, masquerading, denial of service, replaying, and repudiation, so is a mobile agent system. Therefore, issues such as encryption, authorization, authentication, non-repudiation should be addressed in a mobile agent system. Moreover, a secure mobile agent system must protect the hosts as well as the agents from being tampered by malicious parties.

3.2.1 Host Security

In a mobile agent system, hosts continuously receive agents and execute them. Hosts may not be sure where an agent comes from, and are at the risk of being damaged by malicious code or agents (Trojan horse attack). This problem can be effectively solved by strong authentication of the code sources, verification of code integrity, and limiting the access rights of incoming agents to local resources of hosts, such that damages to hosts by malicious agents are limited to the resources available to agents. The solution is realized in the Java security model [Sun99]. Since this problem is effectively solved by the Java security model, it is not of big interest to us.

3.2.2 Agent Security

The main security challenge of mobile agent systems is the protection of agents. When an agent executes on a remote host, the host is likely to have access to all the data and code carried by the agent. If by chance a host is malicious and abuses the code or data of an agent, the privacy and secrecy of the agent and its owner would be at risk.

There can be seven types of attack by malicious hosts [Hoh98b]:

- Spying out and manipulation of code
- Spying out and manipulation of data
- Spying out and manipulation of control flow
- Incorrect execution of code
- Masquerading of the host
- Spying out and manipulation of interaction with other agents
- Returning wrong results of system calls to agents

There are a number of solutions proposed to protect agents against malicious hosts [Tsc99], which can be divided into three streams:

- *Establishing a closed network*: limiting the set of hosts among which agents travel, such that agents travel only to hosts that are trusted, for example, by strong authentication of each host in the network.
- *Agent tampering detection*: detecting whether an agent's execution or data have been tampered with along the journey. Some possible approaches are:
  - Range verification, timing information: the agent and returned results of the agent would be checked against conditions such as a reasonable time period between departure and return, possible ranges of results and so on. This helps to detect very ridiculous modifications of agents and results.
  - Addition of dummy data items and code: dummy data items and code may be added to the agent, so that when the agent is modified maliciously, the dummy items are likely to be modified at the same time, and thus detection of modification can be made easier.
  - Cryptographic watermarks: agents can be designed to implement cryptographic functions such that their result can be proved to be correct.
- *Agent tampering prevention*: hiding from hosts the data possessed by agents and the functions to be computed by agents, by messing up code and data of agents, or using cryptographic techniques.

None of the proposed solutions solve the problem completely. A closed network effectively decreases the chance of an agent being attacked by unknown malicious hosts, however, it also limits the mobility and ability of agents, and hence the openness of the system. Agent tampering detection is possible but requires subsequent efforts to recover from attacks, and is not effective enough for agents that carry out critical missions. Agent tampering prevention would be most effective and useful, but is not yet feasible for arbitrary programs. Most researchers in the area are seeking a better solution, and there is no general

methodology suggested to protect agents. In the mean time, developers of mobile agent systems have to develop their own methodologies according to their own needs.

Apart from attacks by malicious hosts, it is also possible that an agent attacks another agent. However, this problem, when compared with the problem of malicious hosts, is less important, because the actions of a (malicious) agent to another agent can be effectively monitored and controlled by the host on which the agent runs, if the host is not malicious.

3.3 Techniques to Protect Mobile Agents

In the previous subsection, we see a brief overview of the security problems and solutions to mobile agent systems. In this section, we highlight two of the proposed solutions that look most feasible and interesting, namely protected agent states and mobile cryptography.

3.3.1 Protected Agent States [KT99]

In [KT99], three forms of protected agent states are proposed, namely read-only states, append-only logs, and targeted states. They are basically signing and encrypting of agent states based on public key cryptography. Details about public key cryptography may be referred to [Sch96]. The three forms of protection are briefly described here:

- *read-only s*tates: the state (a property of the agent) is digitally signed by the sender of the agent. Therefore, no malicious host can modify the state such that the signature of the state would still be valid.
- *append-only logs*: if a host want to send a message through an agent to the agent owner, but does not want the message to be modified by other hosts, it can digitally signed the message it sends, such that no other malicious hosts can modify this message with the signature still valid.

- *targeted states*: if a host wants to send a message through an agent to the agent owner, it can encrypt the message with the public key of the agent owner, such that the message can only be decrypted by the agent owner.

These protected agent states are simple application of cryptography that helps to achieve the confidentiality and integrity of data of agent. They are effective and feasible to protect agent states, because of the well-established cryptography theory underneath. However, they do not protect the code integrity and confidentiality of agents.

We have adopted and tested these approaches in an experimental system (SIAS). We will discuss this in Chapter 6.

3.3.2 Mobile Cryptography [ST98]

This is a possible approach to protect agent code integrity. The approach works as follows:

If Alice wants Bob to evaluate a function f for her based on Bob's data x, she would encrypt the function f to produced E(f), and implement a program P that evaluates the encrypted function E(f), and send P to Bob. The trick is when Bob runs P, he would not produce plaintext output f(x) that he can read and modify. Instead, he can produce only the encrypted output E(f(x)), which would be readable only by Alice, who has the key to decrypt. Moreover, Bob is unlikely to be able to modify the execution of P, because it implements an encrypted function that Bob would not understand.

This approach seems to be an enlightening way for agent code protection. However, at present, it is proved to be possible for polynomial functions only. On the contrary, there has been an unproven opinion that this approach can be feasible only if the function to be computed can be parameterized. However, if this approach really can extend to other functions, such that arbitrary functions can have an encrypted but executable form that can be evaluated on a remote host, the problem of malicious host will be effectively solved.

While cryptographic techniques provide provable security, in the next chapter, we describe security modeling for mobile agents based on reliability modeling. It does not provide provable security like cryptography does, however, it is targeted for operational security which requires a quantitative measure, rather than analytical security which requires qualitative analysis.

3.4 Reliability Concerns of Mobile Agents

Like other distributed systems, a mobile agent system may fail due to two reasons:

- Site failure: a subset of the hosts in the mobile agent system fails; or
- Communication failure: one or more communication links connecting a subset of the hosts in the mobile agent system fails

In case of a site failure, there can be two different consequences:

- If the mobile agent is not residing on the failing site, the mobile agent keeps alive with its state. However, if the failing site is one of the destinations of the agent, the agent must reroute its itinerary.
- If the mobile agent is residing on the failing site, the mobile agent will be lost. The state of the agent, and computation result will also be lost. Persistence of agents is an issue specific to mobile agent system. However, there is not much new challenge, and existing techniques like logging, check-pointing, and transaction processing may be directly applied.

In case of a communication failure, the mobile agent must be informed of the failure, and it must be able to reroute its itinerary. Otherwise, it will wait indefinitely for the failed communication link to recover, and the system will be virtually dead.

In short, agent persistence and agent rerouting are two of the new challenges that mobile agent systems bring to reliability research.

Chapter 4

Security and Reliability Modeling for Mobile Agents

Throughout the study in the previous chapters, we tried to survey the mobile agent paradigm and technologies, and evaluate the security and reliability issues concerning mobile agents. The approach was, however, intuitive and lacked rigor and formal strength. This is due to the lack of a generic model for mobile agent systems that we can rely on, on which formal calculations, evaluation or proof can be based.

In contrast to the widely accepted "fortress" model for client/server security, there is, at present, no well-known model for mobile agent security. Therefore, in this chapter, we try to emerge with a new model for mobile agent security. We observe that there is a subtle relationship between system security and system reliability, such that we can derive security models from reliability models. We try to derive a security model for mobile agents based on this finding.

Compared to security modeling, reliability modeling for mobile agents is easier. Current reliability modeling techniques can be directly applied to develop the reliability model for mobile agents.

This chapter is organized this way. An attack model of malicious hosts against mobile agents proposed in [Hoh98b] is outlined in Section 4.1 as related work. In Section 4.2 we relate security and reliability, and derive general security models from reliability theories. In Section 4.3 and Section 4.4, we discuss particularly the security model for mobile agents. Finally, in Section 4.5, we present the simple reliability model for mobile agents.

4.1 Attack Model and Scenarios

We find little information about models of mobile agent system and security, especially concerning agent protection. The information fortress model is not applicable to model mobile agents, and the sandbox model is suitable for host protection only. [Hoh98b] is the only work we find that is suitable for modeling agent protection. In this paper, a model of attacks of malicious hosts against mobile agents is proposed. In the model, agents and (malicious) hosts are represented by abstract machines that execute the corresponding agent or attack programs. Abstract machines are modeled using RASPS (Random Access Stored Program plus Stack) machines. An agent RASPS depends on the host RASPS for accessing the environment and communicating with other agent RASPS's. The host RASPS have access to all of the host environment, and is able to read and manipulate the properties of all agent RASPS's running on that host, and control the execution of them. Figure 3 illustrates the model.

This model is plausible because it can be used to describe all the attack scenarios listed in Section 3.2.2. However, this model provides us no quantitative measures for evaluating security solutions, like the failure models being used in reliability theory for evaluating reliability solutions.

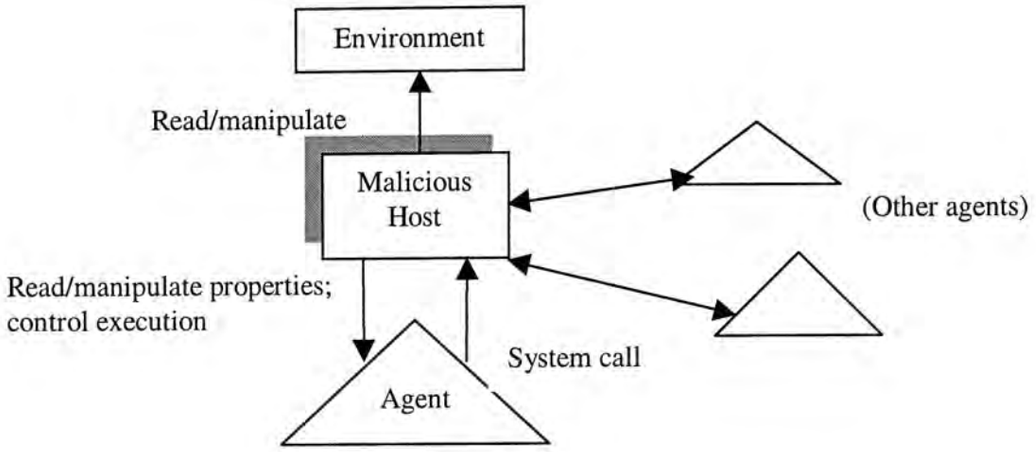


Figure 4.1. Attack model of malicious hosts against mobile agents.

4.2 General Security Models

We realize that a general security model is needed for general system security evaluation, and this model should be applicable to mobile code systems. This is what we find lacking in literature. In this subsection, we try to develop such model.

4.2.1 Security and Reliability

In the broadest sense, security is one of the aspects of reliability [Bir96]. A system is likely to be considered more reliable if the system is more secure. However, the fields of security and reliability follow different paths in development. There has been little effort to unify the results and methods in these two separate fields.

One of the pioneering efforts to integrate security and reliability is [BLOJ94]. In this paper, the authors observed several subtle similarities between the two concerned areas, and draw an analogy of them, which can be summarized in Table 4.1.

Security	Reliability
Vulnerabilities	Faults
Breach	Failure
Fail upon attack effort spent	Fail upon usage time elapsed

Table 4.1. Reliability analogy for security.

This analogy leads us to the intuition that reliability theory, which is well established, can be applied to measure security quantitatively as well. Note the last row on Table 4.1 dictates that the execution time in reliability modeling should be analogous to the attack effort in security. (We would discuss this in Section 4.2.3) Thus, we have *security function*, *effort to next breach distribution*, and *security hazard rate* like the *reliability function*, *time to next failure distribution*, and *reliability hazard rate* respectively, as in reliability theory.

The advantage of this "operational security" approach is that it can give a quantitative measure, *R(e)*, about "how secure a system is" to users. Therefore, users can have an intuitive sense about how much he/she can trust the system being used.

However, this approach does have some drawbacks. Since ultra-high reliability evaluation is so far not very successful, similarly "operational security" based on reliability modeling will not be very desirable for mission-critical applications, contrary to the fact that current security work is focusing on ultra-high security.

Nevertheless, for less mission-critical jobs, such as buying a bunch of flowers from network stores, or ordering a usual dinner from some online service, users would require some certain level of security for the job being carried out, but would not at the level as high as national defense. Users would require an indication about how secure the system is at a relatively low cost [Hoh98c], rather than a pledge of high price that the system is absolutely secure, which may be false.

Therefore, operational security does have its value. If it can be calculated at a low cost, it can be a suitable approach for mobile agents, especially in small-amount electronic commerce. Besides, bringing reliability modeling and security together is also likely to complement the deficiencies in the two fields, in both directions. Current security methods, aiming at ultra-high security, may also help in developing ultra-high reliability methods.

4.2.2 Deriving Security Models

Similar to reliability modeling, if we need to fit the reliability of a system to a particular model, we need to conduct and experiment and collect the required data. There is not much work in this area up till now. One of such is [Jon97], which presents an experiment to model the attacker behavior. The results show that a typical intrusion process can be divided into three phases:

i. the *learning phase,* during which an attacker is spending time to learn the skills for successful attacks (breaches) to the system, and no breach occur at all;

ii. the *standard attack phase*, during which the attacker has acquired some minimum attack skills and "standard" breaches occur. The number of breaches increases with time; and

iii. the *innovative attack phase*, during which the attacker has performed all "standard" attacks, and he must invent new methods for attack, which is more difficult. Therefore, the number of breaches levels off with time.

The number of breaches to a system plotted against the time devoted to attack by an intruder would therefore show an S-shaped curve like Figure 4.2.

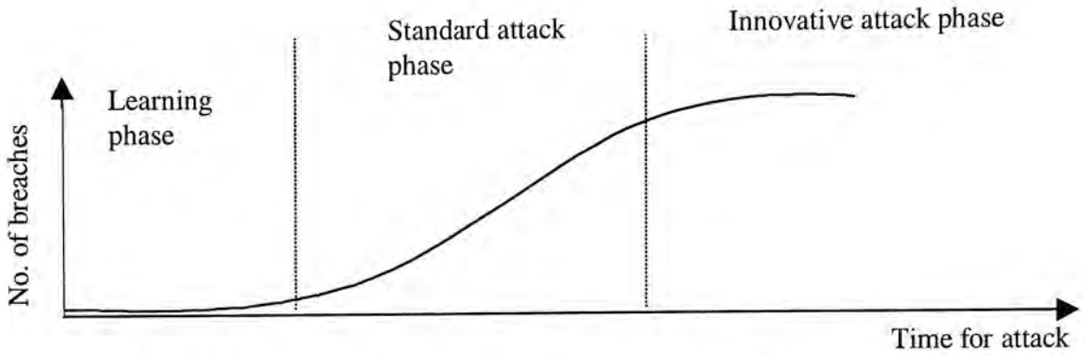


Figure 4.2. Number of successful attacks against time devoted to attack.

This suggests that the intrusion process may fit reliability models such as the *S-shaped model* [YOO83] and *hyper-exponential model* [Ohb84], which also demonstrate S-shaped curves. Furthermore, if there is no learning phase, i.e., the attacker is already very familiar with the system and its vulnerabilities, and does not need to spend time to learn the relevant skills for attack, the curve would become Figure 4.3. Then the intrusion process may fit the *non-homogeneous Poisson process (NHPP) model* [GO79].

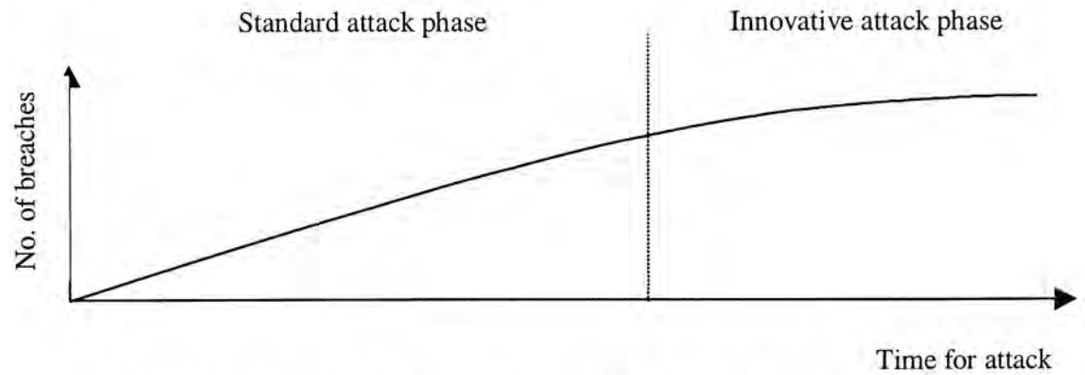


Figure 4.3. Number of breaches against time for attack, without learning.

Moreover, during the standard attack phase, assuming breaches are independent and stochastically identical, the period of working time of the attacker between successive breaches is found to be exponentially distributed (see Figure 4.4). That means the inter-breach occurrence times constitute a Poisson process.

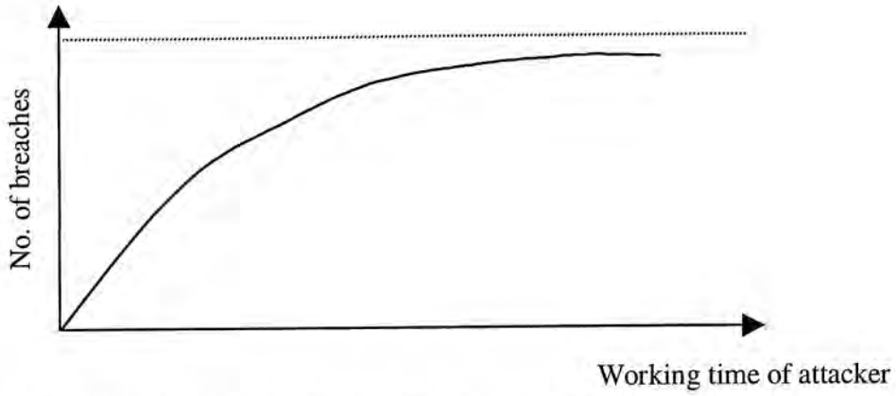


Figure 4.4. Number of breaches against working time of an attacker, during standard attack phase.

This verifies our claim that the intrusion process can be modeled by the S-shaped, hyper-exponential, or NHPP reliability growth models, which all require the number of failures per time period to be exponentially distributed.

There is a point worth noting, though. We are aware of the fact that for these models to hold for modeling reliability, a basic assumption for independent failure must be justified. Similarly, we must also assume that breaches are independent when we adopt these models in the security domain. This is, however, not always the case. Attackers of the same system can form groups to share information and methodology, making it hard to justify the independence of breaches even from different persons. Nevertheless, we assume that attackers do not cooperate, and breaches are independent to each other.

4.2.3 The Time-to-Effort Function

We would model the number of breaches as a function of "effort" spent on attack rather than the usage time elapsed [BLOJ94]. This is because attacker's effort is a more important factor constitutes to a breach, rather than the operational time elapsed.

The working times of attacker in Figures 4.2, 4.3, 4.4 are actually an estimate of the effort spent by the attacker. However, in a normal user's point of view, we would rather have a model of breaches against *calendar time* (i.e., *wall clock*) rather than the effort spent by

attackers. For example, users would be interested to know how long the system would remain free from breaches, rather than how many attackers or how many hours they spend on attack would breach the system, because these are out of users' control. That means, we need a time-to-effort function to capture the effort spent on attack over time for a particular attacker over time.

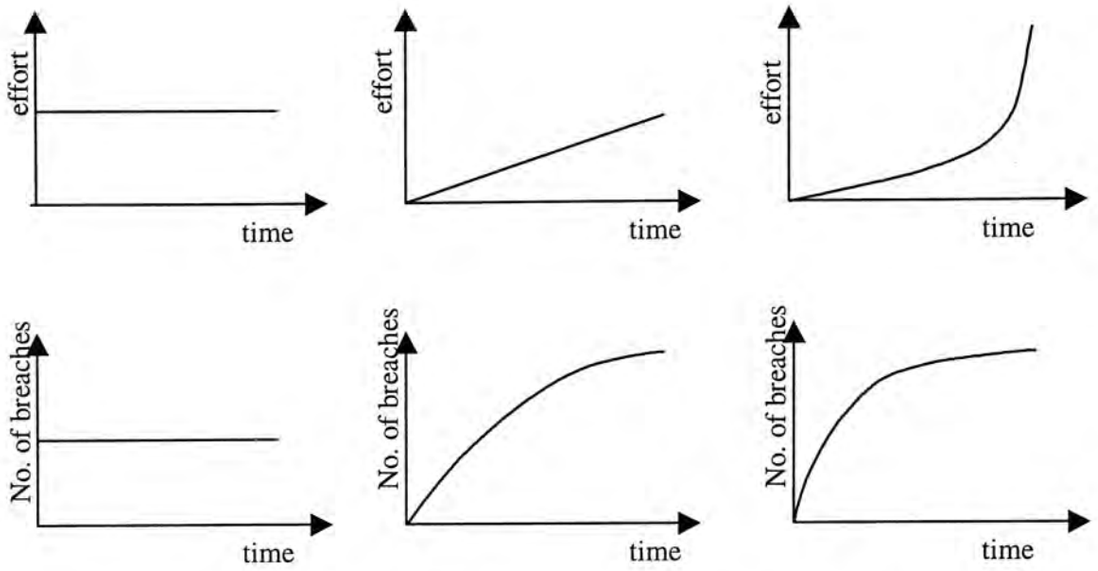
This time-to-effort relationship is, however, difficult to capture, because the term "effort" is actually not very well defined. It may imply the time spent on attack, the number of computers used for hacking, books read, advice from friends, and so on. It is difficult to measure precisely the amount of effort spent. Therefore, we also need to model this time-to-effort relationship. Three simplest models are:

i. *constant effort*: The attacker spends a constant amount k of effort over time. One possible application is that the attacker tries a particular kind of attack only once, at the initial time instance. Furthermore, a constantly zero effort model captures the time-to-effort relationship for a non-attacker, who does not spend any effort for attack at all.

ii. *linearly increasing effort*: The attacker's effort may also increase linearly with time. One possible situation of such is that the attacker may actually be a simple finite state machine running a particular cracking algorithm. The effort spent would then be very largely the number of instructions carried out by such machine. This would be more or less linearly increasing with time.

iii. *exponentially increasing effort*: With the rapid advance of technology, computation speed of both machines and machine owners increase exponentially. Therefore, attackers' effort may also increase exponentially with time.

Figure 4.5 illustrates these simple time-to-effort models, and sketches the corresponding time-to-breach relationship, assuming the exponential effort-to-breach relationship.



a) constant effort b) linearly increasing effort c) exponentially increasing effort

Figure 4.5. Different time-to-effort functions and the corresponding breach-to-time relationship.

We may not be able to derive closed-form solutions for the time-to-breach relationship, though, we can use simulation instead [GLT98a][GLT98b].

4.3 A Security Model for Mobile Agents

Now we develop a simple security model for mobile agents. We focus on the part of agent security against malicious hosts. We derive an operational security model, based on the attack model described in Section 4.1. For simplicity, we have several assumptions to make before developing our model:

1. A mobile agent does not visit any host more than once during a single journey.
2. The attack behavior of any single host is independent of any other hosts. That means, there is no cooperated attack from multiple malicious hosts.
3. A host can attack a mobile agent if and only if it is hosting the mobile agent.

4. The attack behavior of any single host on the agent follows the attack model obtained from [Jon97].

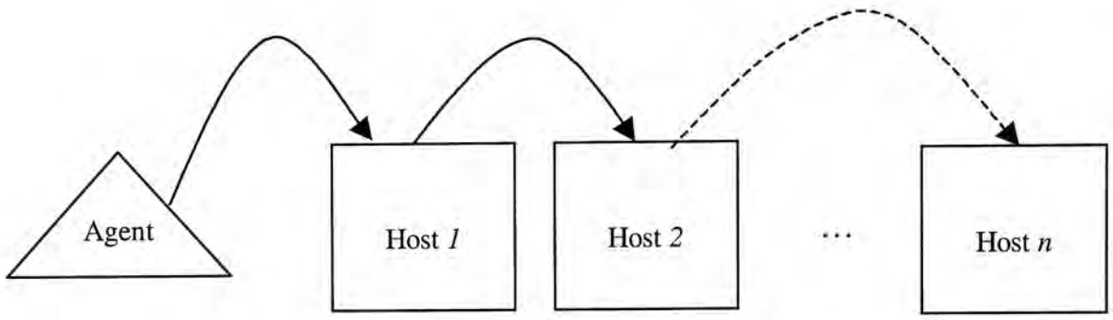


Figure 4.6. A mobile agent traveling along n hosts.

Consider a mobile agent travelling through n hosts on the network, as illustrated in Figure 4.6. Each host, and the agent itself, is modeled as a RASPS machine that does not learn. Therefore, we eliminate the learning phase of attackers as described in Section 4.2.2. Moreover, we assume that the agent does not stay too long on a particular host, such that the agent would leave before the malicious host gets to the innovative attack phase. Therefore, we only need to consider the standard attack phase by malicious hosts.

On arrival to a malicious host, the mobile agent is subject to attack effort from the host. Because the host is modeled as a machine, it is reasonable to estimate the attack effort by the number of instructions for attack carried out, which would be linearly increasing with time. On arrival to a non-malicious host, the effort would be constant zero.

Let the agent arrive at host i at time T_i, for $i = 1, 2, \ldots, n$. Then the effort-to-time function for host i would be

$$E_i(t) = k_i(t-T_i) ,$$

where k_i is a constant. We may call this constant the *coefficient of malice*. The larger the k_i, the more malicious host i is ($k_i = 0$ if host i is non-malicious).

Furthermore, assume that there is no co-operation between malicious hosts, and the $E_i(t)$'s are all independent functions. Therefore, by the Central Limit Theorem, k_i would follow a normal distribution, i.e.,

$$k_i \sim N(\mu_k, \sigma_k^2)$$

Let the agent stay on host i for an amount of time T_i, then there would be breach to the agent if

$E_i(T_i)$ > effort to next breach by host i

Since we assume linear time-to-effort functions, the breach condition is equivalent to

k_iT_i > effort to next breach by host i

or T_i > effort to next breach by host i / k_i

As seen from Section 5.2, it is reasonable to assume exponential distribution of the effort to next breach, so we have

P(breach at host i) = P(breach at time T_i)

= P(breach at effort k_iT_i)

= $1 - exp(-vkiTi)$, v is a constant

= $1 - exp(-\lambda_iT_i)$, $\lambda_i = vk_i$

We may call v the *coefficient of vulnerability* of the agent. The higher the v, the higher is the probability of breach to the agent.

Therefore, the *agent security E* would be the probability of no breach at all hosts, i.e.,

$$E = \prod_{i=1}^{n} e^{-\lambda_i T_i} = e^{-\sum_{i=1}^{n} \lambda_i T_i}$$

4.4 Discussion of the Proposed Model

One possible application of this security function is to calculate the time limit in the Time Limited Black-box security approach [Hoh98a]. We can make estimates of the coefficients of malice k_i's for hosts based on the trust records of hosts. We can also estimate the coefficient of vulnerability v of the agent based on testing and experiments. Therefore, we can calculate the desired time limits T_i's to achieve a certain level of security E.

Conversely, if users specify some task must be carried out on a particular host for a fixed period of time, we can calculate the agent security E for the users based on the coefficients of malice and vulnerability estimates.

However, there are some limitations to this model. Firstly, the validity of the model is based on the assumption of exponential distribution of the effort-to-breach function. This is merely based on the experiment described in [Jon97], and further experiments should be carried out to verify the validity of the model. Secondly, the applicability of the model assumes an effective method to evaluate the coefficients of malice of hosts and vulnerability of agents. For coefficients of vulnerability, it seems possible to find ways to evaluate because there is active research to evaluate software vulnerability [ODK97], and hopefully there will be a good solution. However, for coefficients of malice, it seems infeasible to get a good measure because it must take account of attacker behavior, which is difficult to model.

One possible method to work around the evaluation of the coefficients is to let users evaluate different hosts and agents in an open environment. This is similar to the mutual trust evaluation in some electronic bidding systems. Assuming the majority of users are rational, this is a possible solution. One of our future works is to develop our model in this sense.

4.5 A Reliability Model for Mobile Agents

Unlike security modeling, software reliability modeling theory has been developed for years, and turns out to be a successful research area. Mobile agents are merely pieces of software

when they are running on a particular host. Therefore, traditional reliability modeling can easily apply. In this section, we derive a simple model for evaluating the reliability of a mobile agent system.

All software may fail over time of usage due to bugs or defects in the software. In general, the failure rate of a software system grows, unlike a hardware system, over time before it becomes stable, due to testing and debugging of the software. Different software systems demonstrate different growth pattern, and can be captured by different growth models, for example, the S-shaped, NHPP, and hyper-exponential growth model discussed in Section 4.2.

For simplicity, we assume a mobile agent system already in use for a period of time, and which has become stabilized. That means, we are going to assume *a constant reliability function* for a mobile agent on each host. This also implies that the failure rate of an agent on each particular host is constant

Consider the same scenario in Figure 4.6, where an agent travels around n hosts. Suppose the failure rate of the agent running on host i is p_i, then the *failure rate of mobile agent system with n hosts*, P, assuming no communication failure, would be

$$1 - \text{P(no failure of agent on each host)} = 1 - (1-p_1)(1-p_2)\ldots(1-p_n)$$

If all hosts are homogeneous, the failure rates at all hosts are likely to be the same, i.e., $p_1 = p_2 = \ldots = p_n = p$, then $P = 1 - (1-p)^n$.

If we plot this failure rate P against the number of hosts n, the curve would be of the form like Figure 4.7.

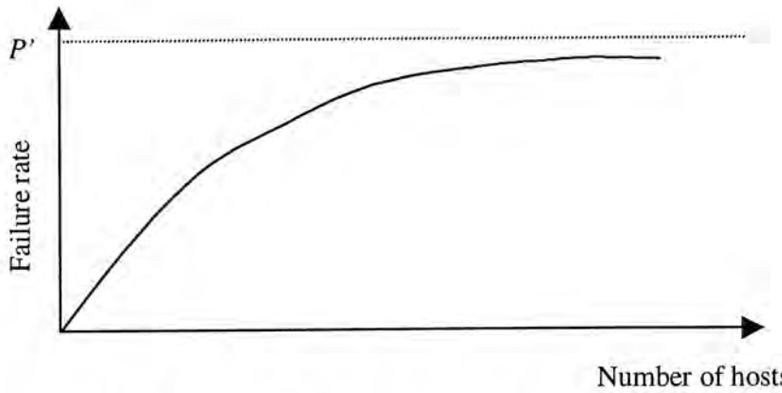


Figure 4.7. Failure rate of mobile agent system Vs number of hosts

This can be called the failure rate curve of the mobile agent system described. Reliability of the corresponding mobile agent system can be defined as the probability of no failure. Therefore, it is simply *(1 - P)*. Therefore, the curve in Figure 4.7 can also be called the *reliability curve* of the system.

From the figure, we see that there is an upper bound for the system failure rate. Theoretically this rate is one. However, in practice, the failure rate of a software system should never grow up to as high as one. If, as the number of hosts increases to a certain maximum value, the curve saturates at a particular value *P'*, we take this value to be the *saturated failure rate* of the mobile agent system.

With this reliability model, it follows directly that a system with a low value of P' is more reliable than one with a high value of P'. Later in Chapter 6, we will apply this model to evaluate some reliability measures we developed.

Chapter 5

The Concordia Mobile Agent Platform

In Chapters 3 and 4, we discussed the theory side of mobile agents. In this chapter and the next, we will switch to the implementation side of mobile agents. This short chapter describes the Concordia mobile agent platform. It is here to facilitate the discussion of the next chapter, in which a system will be built based on this platform.

Most details of the Concordia mobile agent platform can be found in [Mit99a]. Instead of paraphrasing everything from the source, in this chapter, we only give a brief overview of the platform, and compare this platform to others like Aglets, highlighting some special features of Concordia.

5.1 Overview

Concordia is a Java-based framework for mobile agent application development. A simplest Concordia system consists of a Java Virtual Machine (JVM), a Concordia server, and at least one mobile agent on a network node.

The Concordia server is a Java based program that runs in the JVM. A mobile agent in the Concordia framework is a Java object managed by the Concordia server. A user can

create a mobile agent using the Java language and the Concordia Application Programming Interface (Concordia API). The user can program a mobile agent such that the agent moves to different network nodes to perform tasks on different computers. The Concordia server invokes a mobile agent created to perform the tasks programmed by user. If an agent is programmed to move to another network node, the Concordia server suspends the agent and sends it to the destined node.

The Concordia server also listens on a particular network port of the local machine for arrival of incoming mobile agent request. If there is a request from another network node for receiving of an incoming agent, it creates a new agent, copying all the states of the agent, and resumes execution of the agent. After that, the sending Concordia server can erase the copy of the mobile agent on the original host. In this way, the mobile agent *virtually* travels through the network from one server to another, and performs tasks on different hosts.

5.2 Special Features

Compared with other mobile agent framework, Concordia has the following features:

- It is Java-based. The running time of the system may be a little bit slower than those non-Java-based framework such as D'Agents. However, it also enjoys the advantages of Java, such as object-orientation, modularity and code reuse.
- It has a simple application programming interface. Programming an agent in Concordia is a relatively easy task, compared with other Java-based system like Aglets. Basically, a Concordia programmer needs only to define a mobile agent object inheriting from the Agent class provided by the Concordia API. The Concordia handles most of the life cycle services of agents. On the other hand, with Aglets, the programmer must take care of the life cycle of an agent.

- It allows the administrator of the Concordia server to manipulate directly the execution and itinerary of a mobile agent. This is a preferable feature when we need to simulate malicious host actions.
- It does not facilitate direct messaging between mobile agents. It is less powerful than Aglets in this aspect. On the other hand, Aglets support direct message-passing between applications and agents, and between agents and agents
- From our experience with Concordia (evaluation version), to our surprises, it fails frequently. We cannot find out the reasons of failure. Probably, it is due to bugs in the Concordia server program.

The materials presented above are far from being a complete introduction to Concordia. However, for the purpose of the discussion of this thesis, they are most, if not all, the necessary information to proceed forward. In the next chapter, we will discuss SIAS, an electronic commerce application built on top of the Concordia framework.

Chapter 6

SIAS: A Shopping Information Agent System

This chapter presents *SIAS*, the *Shopping Information Agent System*. In contrast to Chapters 3 and 4, this chapter is dedicated for an implementation-wise and application-specific discussion of mobile agent security and reliability issues. The aim of this discussion is to complement the previous discussion by giving more focus on the security and reliability problems specific to a particular application. The particular application chosen is a shopping information system, which is an essential part in electronic commerce. As stated in Chapter 1, this discussion does not necessarily correlate with the discussion in Chapter 4. However, the reliability model developed in Chapter 4 serves to be a handy tool to evaluate the reliability of the application.

6.1 Goal and What the System Does

SIAS implements mobile agents to retrieve product information in an *electronic market* for users. An electronic market consists of hosts that sell products on the network. Each seller maintains a database that stores the prices and quantities in stock of different products available at that host.

SIAS keeps a roster of all hosts in the electronic market and a list of all products available in the market. It allows users to specify a set of products and the corresponding quantities they want to buy from the list. An agent is created for the user who has specified the list of products and quantities. The agent, on behalf of the user, will collect information about availability and price from hosts in the network. The path of the agent is determined before the agent is launched, according to the roster of hosts kept by the system. After the agent visits all hosts specified in its itinerary, it returns to its sender and reports the lowest prices and corresponding sellers. The design of the system is described in details in the next subsection.

Rather than building a "toy system", the ultimate goal of building SIAS is to push it at the end as real an application as possible. Therefore, security and reliability issues must be very carefully considered. In the following sections, we discuss the system design, implementation, and security and reliability considerations respectively.

6.2 System Design

SIAS is designed using the object-oriented paradigm because the concept of objects is useful to describe agents. There are three main types of objects in the system, namely Agents, Launch Servers and Database Servers. We describe the object details and control flow of the system in this subsection.

6.2.1 Object Description

The three objects are designed as follows:

- The *Agent* object: it keeps a list of product identification numbers (IDs) and a list of the corresponding quantities specified by users. It is responsible to travel around the network and collect product information for users from different hosts.

- The *Launch Server* object: it is responsible for creating agents for users, sending the agents to the network, and receiving the agents when they finish visiting all the hosts specified in their itineraries.
- The *Database Server* object: it stores the information of products available at a particular host, (each host has its own instance of this object) and is responsible for retrieving required information for an agent when it arrives to the host.

Figures 6.1, 6.2 and 6.3 show the details of the objects respectively.

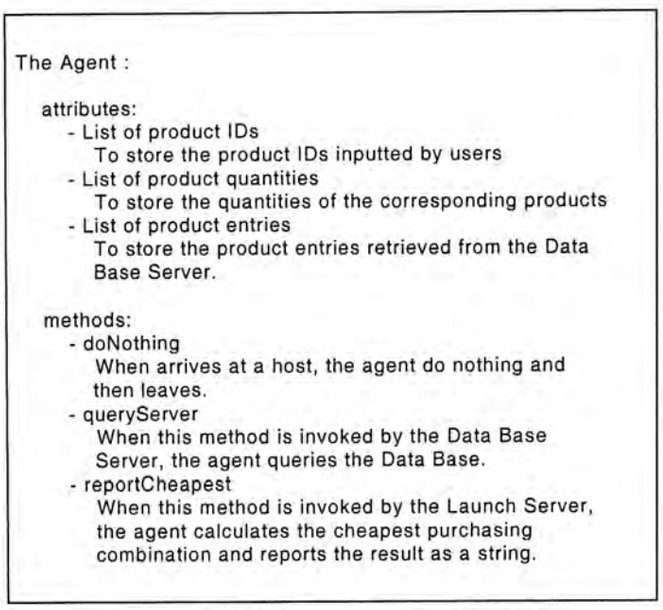


Figure 6.1. Object details of Agent.

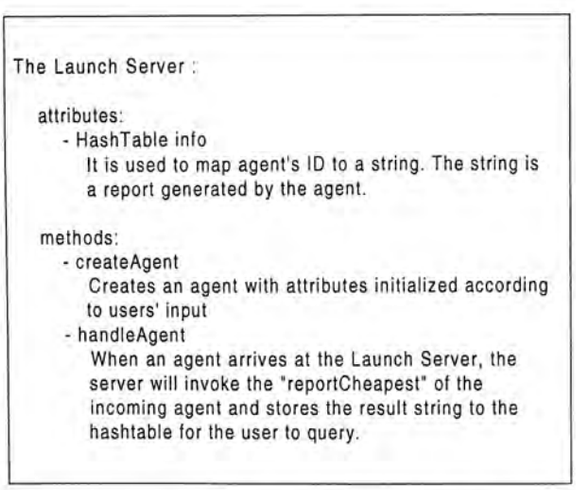


Figure 6.2. Object details of Launch Server.

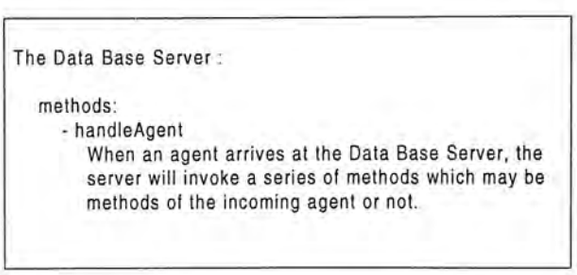
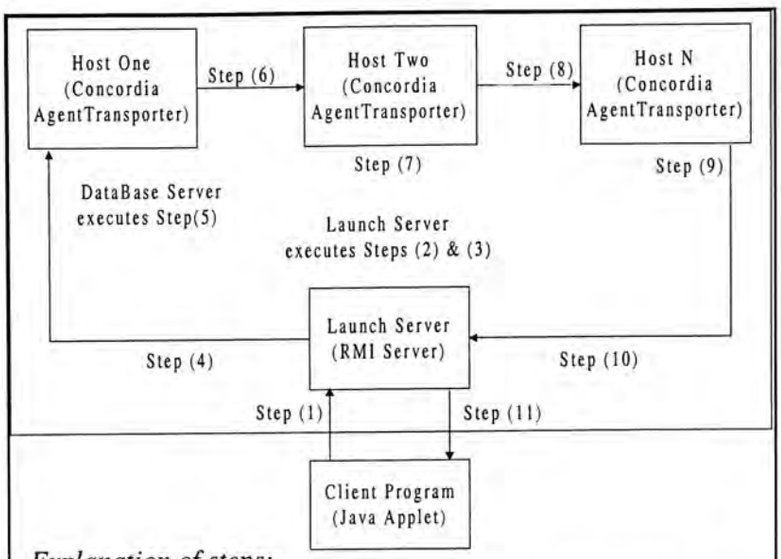


Figure 6.3. Object details of Database Server.



Explanation of steps:

1. *Client program launches a request to the Launch Server object upon user input using Java Remote Method Invocation (RMI);*
2. *Launch Server creates an Agent object;*
3. *Launch Server initializes the agent with user-specified products and quantities, and the itinerary of agent;*
4. *Launch Server sends the agent to the network;*
5. *Database Server on Host One retrieves the required information for the incoming agent;*
6. *Agent goes to the next destination;*
7. *Database Server on Host Two repeats Step (5);*
8. *Agent goes to other hosts in the itinerary;*
9. *Database Server on each host repeats Step (5);*
10. *Launch Server receives the returning agent and calculates the cheapest purchasing combination;*
11. *Launch Server reports the cheapest purchasing combination to client program by Java RMI.*

Figure 6.4. Control flow of SIAS.

6.2.2 Flow Description

When user makes a request for product information, an Agent is constructed with the product and quantity lists initialized properly by the Launch Server, and the agent will start its tour on the network. Whenever it reaches a host with a Database Server, it stays there, collects information of user-selected products, and then goes to another host. When it has visited all the hosts that are specified in its itinerary, it will calculate the lowest prices, and finally reports to user. The detailed control flow of the system is illustrated in Figure 6.4.

6.3 Implementation

SIAS is implemented using the Java programming language with the support of the Concordia API [Mit99a]. The choices of programming language and supporting API, together with some other implementation details, are discussed in this subsection.

6.3.1 Choice of Programming Language

Java is chosen to be the programming language for implementation of SIAS with two main reasons, apart from its object-orientation and portability features.

- First, most mobile agent APIs currently available, including Concordia and Aglets [IBM99], are built on top of Java.
- Second, Java provides an API that helps us to implement security measures for our system.

6.3.2 Choice of Mobile Agent Platform

The Concordia mobile agent API is chosen, among others like IBM Aglets Software Development Kit (ASDK), because it is simple and easy-to-use. This saves us a lot of time from developing the system. However, communication between agents would be difficult to implement with Concordia, yet it does not affect our choice because there is little communication between agents in SIAS.

Another important point in choosing Concordia is that it allows easy manipulation of execution of agent codes. Therefore, we can simulate a malicious host that does not execute an agent in the intended way easily.

6.3.3 Other Implementation Details

Agent objects are instantiated by the Launch Server object. The Launch Server object fills the product list and quantity list of the created agent, determines the itinerary of agent and then sends the agent out to the network.

Referring to Figure 6.4, there is an object on each host called AgentTransporter. This is introduced by the Concordia API, and it is responsible to listen for incoming agents (see Chapter 5). When an agent arrives, the AgentTransporter raises an event signal, and invokes the Database Server or Launch Server to handle the agent. The Database Server use Java Database Connectivity (JDBC) to handle the connectivity between agents and the database that store the product information at each host.

6.4 Snapshots

We have implemented a graphical user interface (GUI) for SIAS. We present some screen shots that demonstrate the use of SIAS in this subsection. Figures 6.5, 6.6 and 6.7 show the start-up page and GUI of SIAS.

Figure 6.5. The starting page of SIAS: clicking on the text "Let's Go Shopping" will bring out a login window.

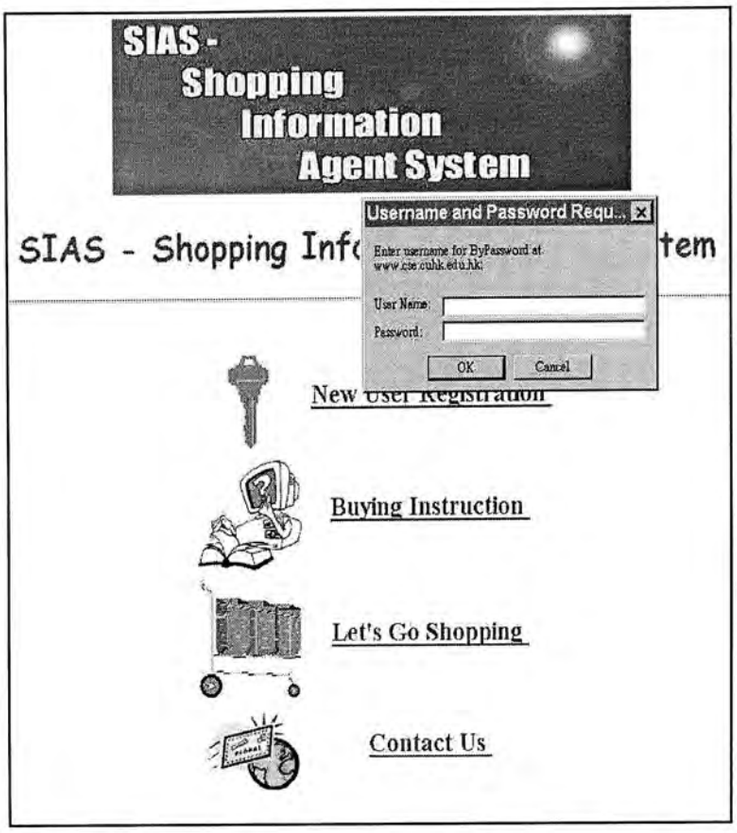


Figure 6.6. The login window: SIAS is a password-protected system. In order to log into the system, a correct login name and the corresponding password must be given.

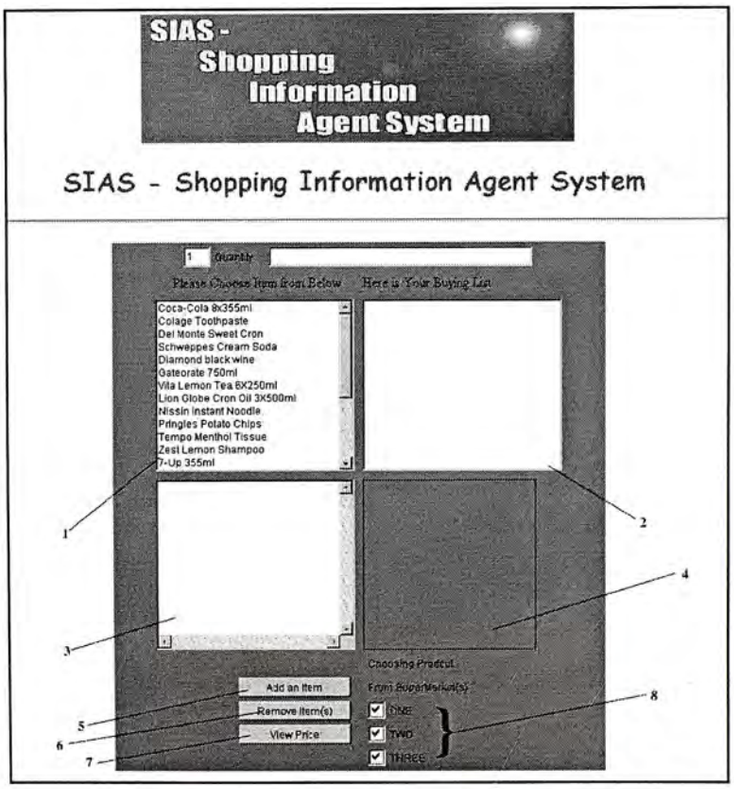


Figure 6.7. The system starts up, showing the user interface of the system.

The numbered items in Figure 6.7 (the GUI of SIAS) are described here:

1. Item List: this list contains a list of all products available in the market. User can choose the products they want from it.
2. Buying List: this list contains a list of products that user has chosen.
3. Description Text: this text area displays a description of the product, such as the weight and ingredients.
4. Photo Displaying Area: this area displays a photo of the selected product.
5. Add Item Button: this button is used to add a selected item from Item List to Buying List. Users can also add a selected item to the Buying List by double clicking on the Item List.
6. Remove Item Button: this button is used to remove a selected item from the Buying List.
7. View Price Button: this button is used to invoke the Launch Server, create an agent, and query the price s of products listed on the Buying List.
8. Check Box Group: this group of check boxes allows users to select the stores that users want the agent to visit and query.

Figure 6.8 and Figure 6.9 show a typical run of SIAS.

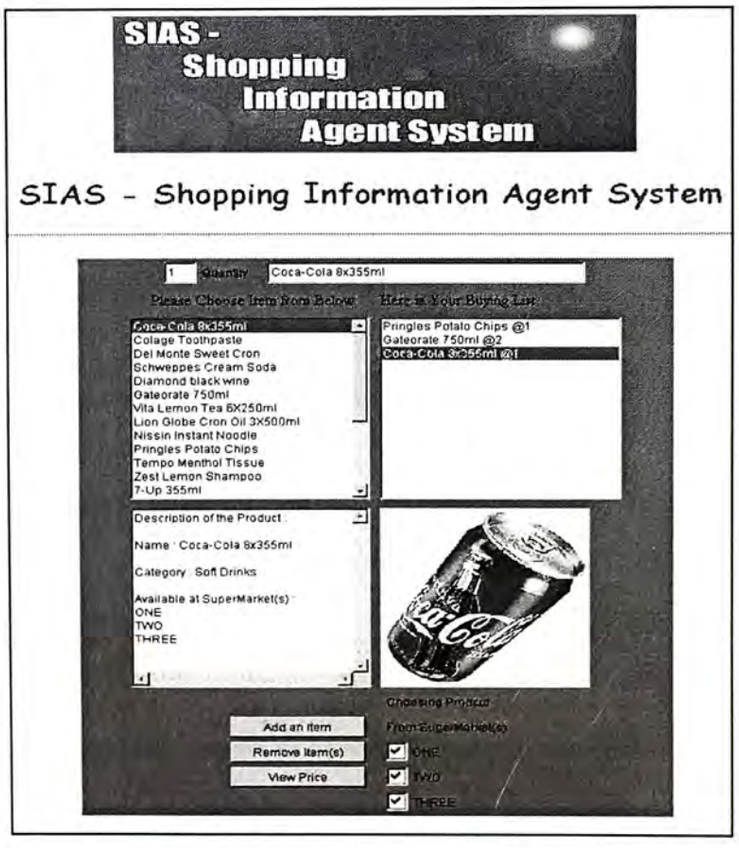


Figure 6.8. User is choosing products.

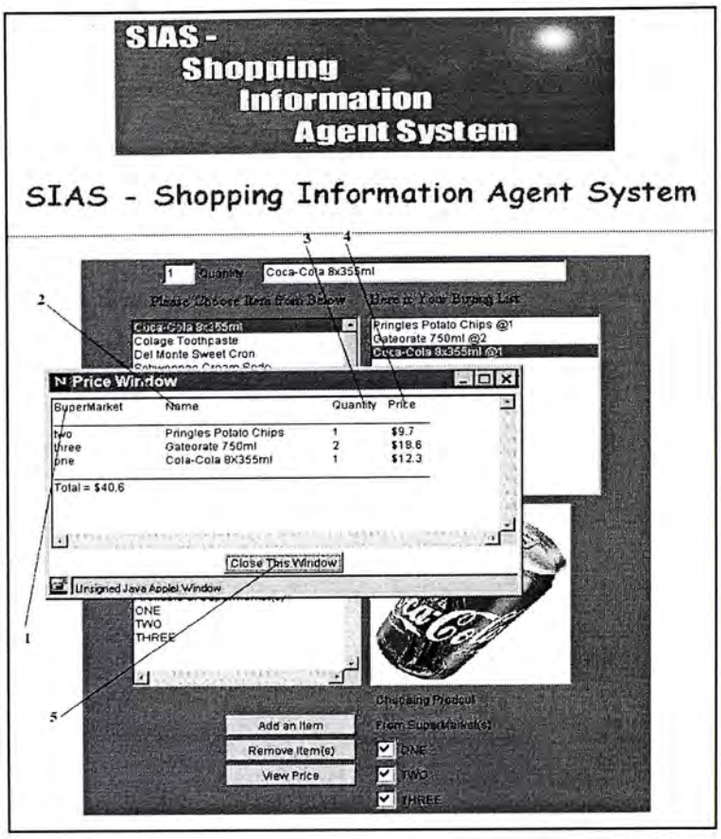


Figure 6.9. The system reports the query result in the Price Window.

The numbered items in the report window in Figure 6.9 is described here:

1. Supermarket column: this column displays, for each product, the store that is selling at the lowest price.
2. Name column: this column displays the name of each product.
3. Quantity column: this column displays the quantity of each product that users have specified.
4. Price column: this column displays the price of each product at the quantity specified by user.
5. Close Window Button: this button is used to close the report window.

6.5 Security Design of SIAS

SIAS is a web-based system, attacks from the Web to the system are likely, and security is an important issue of the system design. Moreover, system security is of crucial importance

to applications in an electronic marketplace, where money transaction is concerned. This section describes the security challenges of SIAS, and presents a simple but original approach to solve the problems.

SIAS is a mobile agent system, and is therefore subject to all kinds of attacks described in Chapter 3. Both host security and agent security would be issues of SIAS. However, since we have built SIAS using the Java programming language, which provides strong security mechanisms to protect hosts against malicious programs or agents through the use of Java Virtual Machine (JVM) and sandbox, the host security problem is very much simplified and solved. On the other hand, agent security needs much more concerns. In what follows, only agent security of SIAS against malicious hosts would be discussed.

6.5.1 Security Problems of SIAS

We start our discussion by giving a set of security requirements for SIAS. There are three primary requirements:

1. *Integrity*: the query results reported by an agent must truly represent the market prices of the products and at the quantities specified by the user.
2. *Confidentiality*: information collected from a store by an agent should not be revealed to other hosts or agents.
3. *Authenticity*: an agent must visit and collect information truly from the list of stores specified by users.

Without special design, all these requirements can be violated by actions of a malicious host. There are four possible types of such attacks to agents that can compromise the security of the system, namely modification of the query products of an agent, modification of the query quantities of an agent, spying out and modification of query results, and modification of the itinerary of the agent.

- **Modification of query products**

The list of products specified by user is stored as the product ID list attribute of an Agent object, in plain text form. When an agent goes to a malicious host, the malicious host can change the product list the agent wants to query. When the agent later go to another host, the later host will respond to the changed products of query and report wrong information. This violates the integrity of the queries.

- **Modification of query quantities**

Similar to the modification of query products, when an agent goes to a malicious host, the malicious host can change the quantities of products the agent want to query, which is simply in plain text form. When the agent goes to another host, the later host will respond to the modified quantities of query, and report wrong information. This also violates the integrity of queries.

- **Spying out and modification of query results**

Agents carry query results also in plain text form. Therefore, when an agent goes to a malicious host, the malicious host can spy out and modify the results that the agent has collected from previous hosts in such a way that the changed results would favor the malicious host itself. For example, a malicious host may raise the prices quoted by other hosts, to convince the user that it is selling at the lowest price, which is not true. This violates the confidentiality and integrity of query results

- **Modification of itinerary of an agent**

The itinerary of an agent is accessible to hosts that have control over the Concordia platform where the agent lands and executes. When an agent goes to a malicious host, the malicious

host can modify the path of the mobile agent so that the agent will go to a host not specified by user. This violates the authenticity requirement of the system.

The above attacks are only a subset of possible attacks. There are other attacks such as replaying of query results and masquerading of hosts. However, these attacks are more complex, and require more efforts for both attack and defense. For the time being, we consider the four simple attacks only.

6.5.2 Our Solutions to the Problems

Having figured out the four system vulnerabilities described above, we have to implement mechanisms to protect our systems against exploitation of these vulnerabilities. As stated in Chapter3, there is currently no good solution to mobile agent security in general. Therefore, we have to devise our own mechanisms to defend against possible attacks.

We develop a simple approach to protect agents in SIAS against attacks from malicious host, based on protected agent states (see Chapter 3). It is actually a hybrid approach of the solutions, i.e., *establishing a closed network, agent tampering prevention and agent tampering detection*, discussed in Chapter 3.

- *Closed network*: we introduce a new object, namely key server or KeyServer, into our system, which provides a public key infrastructure for agents and hosts in the system. Each agent or host should have a public key certificate registered to the key server for encryption or decryption purposes later on. The Launch Server generates a pair of keys for each agent created, and registers the public key of the agent with a unique agent identification number to the key server at run-time. On the other hand, each host must identify itself and register its public key to the key server before, by such means as a formal paper writing. This in effect establishes a closed set of hosts registered and

known to the key server. Agents are then confined to travel among a closed network form by these hosts.

- *Agent tampering prevention*: to protect query integrity, an agent can digitally sign its list of products and quantities using its private key, before it is launched. A host receiving the agent should verify the product and quantity lists with the signatures. Since only the Launch Server possess the private key for the agent, malicious hosts would not be able to fake the signature of the product and quantity lists.

  Moreover, each host should encrypt the query results returned to the agent with the public key of the agent. Therefore, only the Launch Server can decrypt the query result, and confidentiality of query results is achieved. Furthermore, each host should digitally sign the query result it provides to the agent to ensure integrity and authenticity of the query result returned.

- *Agent tampering detection*: the itinerary of an agent is an variable hidden by the Concordia system and normally not accessible. However, hosts can actually have access to the itinerary of an incoming agent by controlling the execution of the Concordia agent transporter. A malicious host would be able to change the itinerary of the agent. As before, the straightforward method of protecting the itinerary is to encrypt it. However, this requires modification of the agent transporter of Concordia, which is not desirable to us.

  We work around the problem by making the itinerary of an explicit attribute of an agent. When an agent arrives at a host, the host should read the itinerary of the agent, and encrypt the itinerary using its own private key to form encrypted itinerary EI_1. Then when the agent arrives at a second host, the second host should encrypt, with its own private key, EI_1 concatenated with the itinerary it reads from the agent. This keeps on to

form a chain of encrypted itineraries. When the agent returns to the Launch Server, the Launch Server will decrypt the chain of encrypted itineraries using the public keys of the hosts to check the consistency of all itineraries and check with a copy of the original itinerary it saves before launching the agent. If a malicious host ever changes the itinerary of the agent, it is likely to be reflected in the encrypted itinerary chain and detected finally.

Figure 6.10 illustrates the changes introduced to SIAS for the security solutions described above, and Figure 6.11 illustrates the control flow of security-enhanced SIAS. Note that the encryption algorithm chosen is the most common RSA algorithm [RSA78]. These changes happen to be very similar to Karnik's idea [KT99] of "read-only states", "targeted states", and "append-only logs". In fact, these are only straightforward application of cryptographic techniques.

I. {Product ID list} *changed to*:
{Product ID list}•sig_A({Product ID list})

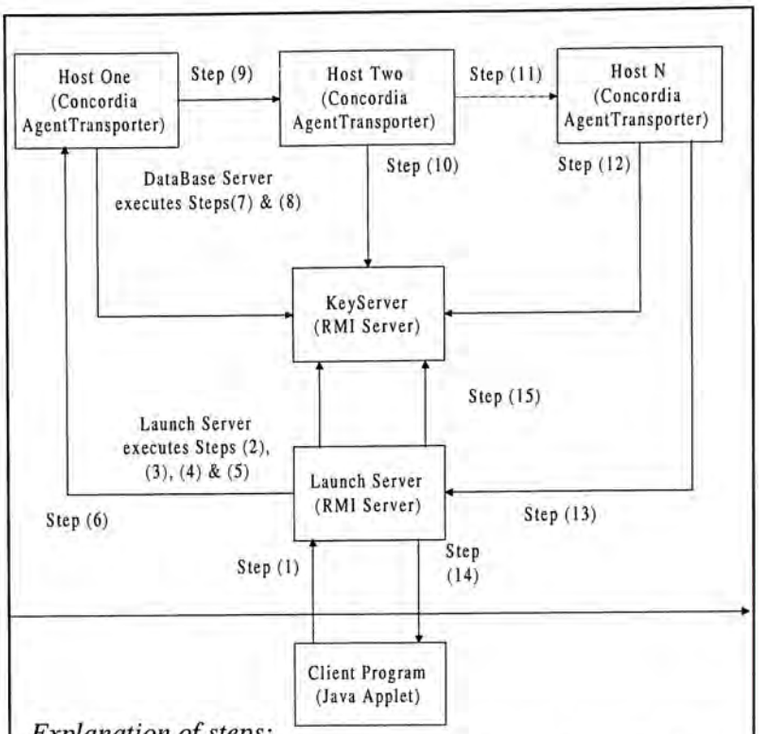
II. {Product Quantity list} *changed to*:
{Product Quantity list}•sig_A({Product Quantity list})

III. {Query result} *changed to*:
D_A({Query result}•sig_H({Query result}))

IV. *New attribute (chain of encrypted itineraries)*:
$E_{HN}(E_{H(N-1)}(\ldots E_{H2}(E_{H1}$(Itinerary at Host 1) • Itinerary at Host 2) • …
Itinerary at Host N-1) •Itinerary at Host N)

Key
A: agent;
H: host;
H(k): k-th host visited by the agent;
$sig_X(Y)$: digital signature of Y using the private key of X;
$E_X(Y)$: the ciphertext of Y encrypted by the private key of X;
$D_X(Y)$: ciphertext of Y encrypted by the public key of X.

Figure 6.10. Changes introduced to secure SIAS



Explanation of steps:

1. *Client program launches a request to the Launch Server object upon user input using Java Remote Method Invocation (RMI);*
2. *Launch Server creates an Agent object;*
3. *Launch Server initializes the agent with user-specified products and quantities, and the itinerary of agent;*
4. *Launch Server generates a key pair for agent;*
5. *Launch Server signs the product and quantity lists for agents and registers the public key of agent to Key Server;*
6. *Launch Server sends the agent to the network;*
7. *Database Server on Host One retrieves public key of agent from Key Server, and verify the signatures of product and quantity lists of agents*
8. *Database Server retrieves the required information for the incoming agent, signs the results using its own private key, and encrypt the results using the public key of agent, and also starts the chain of encrypted itineraries for agent;*
9. *Agent goes to the next destination;*
10. *Database Server on Host Two repeats Steps (7) & (8);*
11. *Agent goes to other hosts in the itinerary;*
12. *Database Server on each host repeats Steps (7) & (8);*
13. *Launch Server receives the returning agent and calculates the cheapest purchasing combination;*
14. *Launch Server decrypts the query results, and verifies the signatures of the query results. It also detects change of agent itinerary by decrypting the chain of encrypted itineraries, and finally reports the cheapest purchasing combination to client program.*
15. *Launch Server deletes the public key entry of the finished agent from the key server.*

Figure 6.11. Control flow of security-enhanced SIAS.

6.5.3 Evaluation of the Secure SIAS

In this subsection, we evaluate the security design we implemented in previous subsection. There are two aspects to evaluate. First, we analyze the security provided to SIAS by the additional measures. Then, we measure the performance overhead introduced to the system by such measures, according to different sizes of query carried by an agent, and also to different number of hosts in the system.

6.5.3.1 Security Analysis

The security of the additional measures lies mainly on the introduction of a key server that facilitates the use of public key cryptography. Assuming the key server and the communication channel with the it are secure enough, which can be justified by the popularity of Kerberos [MIT99b] and Secure Socket Layer [FKK96], the closed network we want can be built effectively.

Furthermore, if the keys of agents are managed properly, the prevention of modification of the signed product and quantity lists of an agent by a malicious host is supported by the security of the RSA encryption algorithm, of which the difficulty to break is equivalent to the factoring problem. The time complexity for breaking the system depends on the length of the key in number of bits. The longer the key is, the more secure would be the system. In our implementation, we have chosen a key length of 128 bits. This would be sufficiently secure for domestic purpose.

Similarly, a malicious host would understand or modify the encrypted query results collected by an agent from another host at the same complexity. Therefore, integrity of queries, and confidentiality and integrity of query results, as described in Section 4, can be achieved by prevention of tampering.

For the detection of modification to itinerary of an agent by a malicious host, suppose there is only a single malicious host, out of N hosts, that wants to modify the

itinerary of an agent. Since the encrypted itineraries are chained together, with one encapsulating another, the malicious host would need to fake all the (N-1) encrypted itineraries from other hosts to avoid being detected, which would be too complex to an ordinary attacker. Therefore, the itinerary of the agent can be assured, and authenticity achieved.

However, as mentioned in Chapter 3, there do exist other attacks that we have not considered completely, such as replaying attacks, timing attacks, and repeated cipher-text attacks. Protection against these attacks would be a direction for future work on SIAS.

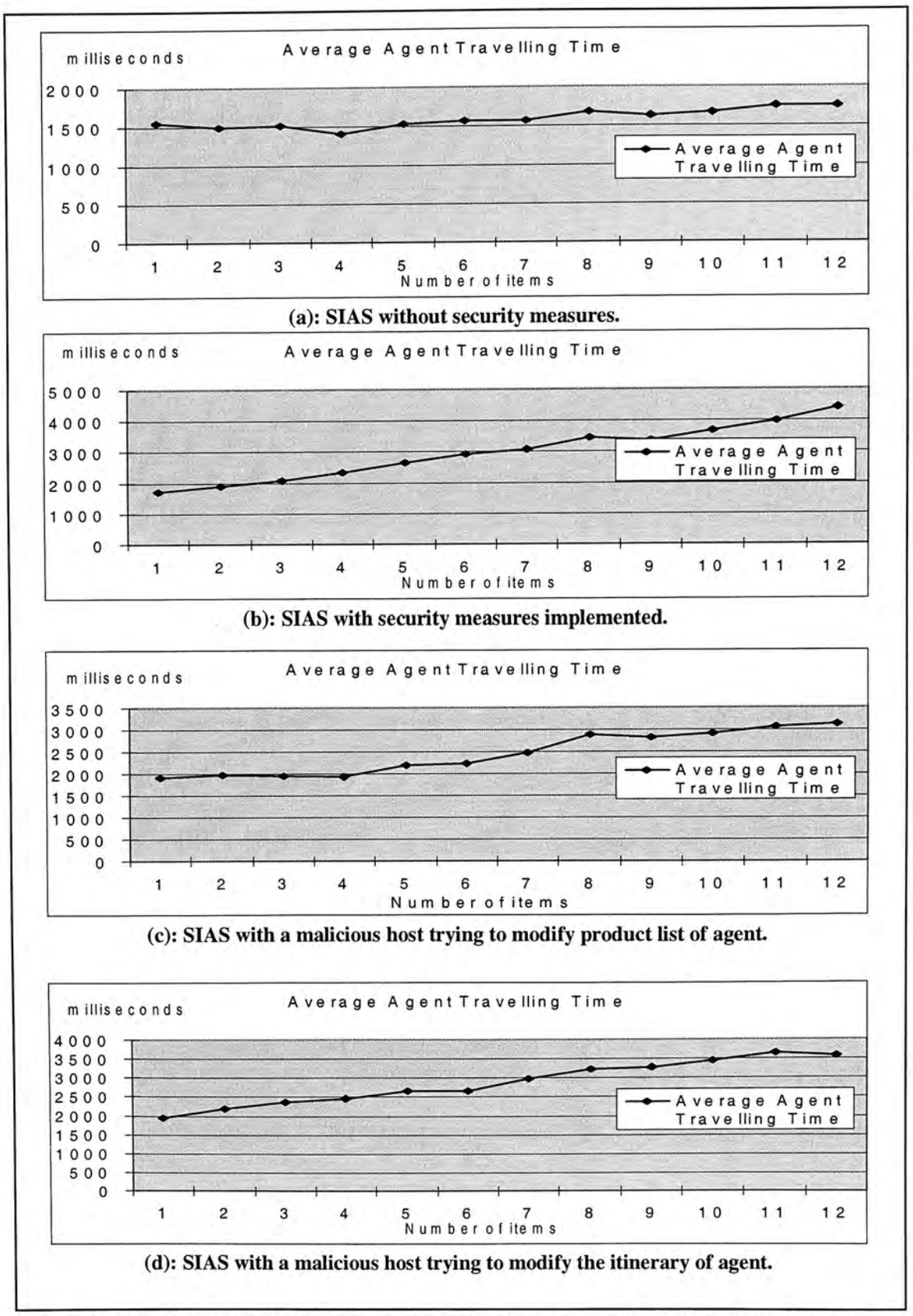
6.5.3.2 Performance Vs Query Size

We have tested the times for SIAS to launch a single agent before and after implementation of the security mechanisms described in Section 4. Round trip times (RTTs) required for an agent to travel around an electronic market, consisting of three hosts, are measured under different situations. Queries of different sizes (number of product items) have been tested. RTTs measured are plotted against the query sizes in Figure 6.12.

Figure 6.12(a) shows the results for the SIAS implementation without security measure implemented. The RTT increases very slightly with the size of query. The overhead introduced by each additional item in average is only about 250/6 = 41.7 milliseconds. This can be explained by the small change in delay of database query with different query sizes.

On the other hand, Figure 6.12(b) shows that for the security-enhanced SIAS, the RTT increases very fast and linearly with the size of query. The overhead introduced by each additional item of query is about 250 milliseconds, which is about six times the overhead of the system without security measure. This can be explained by the extensive use of the RSA algorithm to encrypt and decrypt each item, which is time consuming, especially when the key is long. However, a longer key gives stronger protection to the system. Therefore, we see a trade-off between performance and security for SIAS.

In addition to measuring the performance overhead introduced by the security measures, we also simulate malicious hosts trying to modify the product list and itinerary of an agent in SIAS, and measure the overheads introduced by the actions of malicious hosts.



(a): SIAS without security measures.

(b): SIAS with security measures implemented.

(c): SIAS with a malicious host trying to modify product list of agent.

(d): SIAS with a malicious host trying to modify the itinerary of agent.

Figure 6.12. Round trip time measurements for an agent in SIAS with different configurations.

The results are reported in Figures 6.12(c) and 6.12(d).

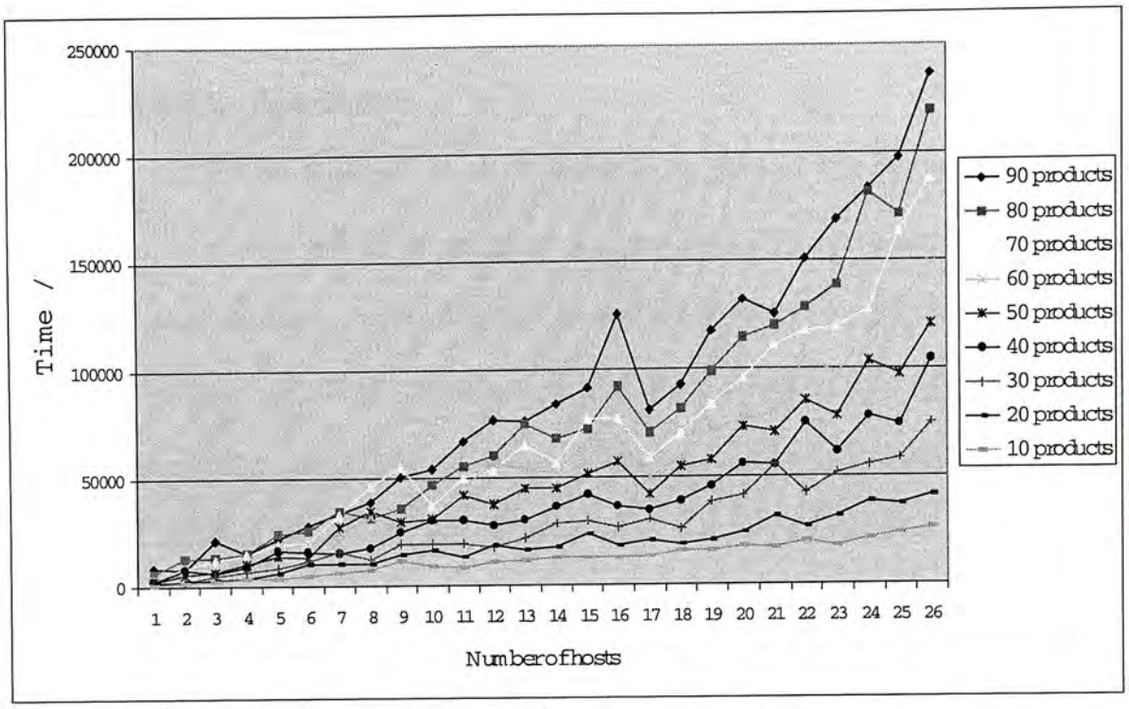
Both graphs show that an agent takes more time to travel around when there is attack from malicious host, compared with the measurements in Figure 6.12(a). The RTTs in (d) is slightly larger than those in (c) in general, because agent itinerary is actually an internal property of an agent, and it takes the malicious host extra time to access the itinerary. The delays of agents by malicious hosts suggest that the agent round trip time may also be used as a measure for tampering detection.

6.5.3.3 Performance Vs Number of Hosts

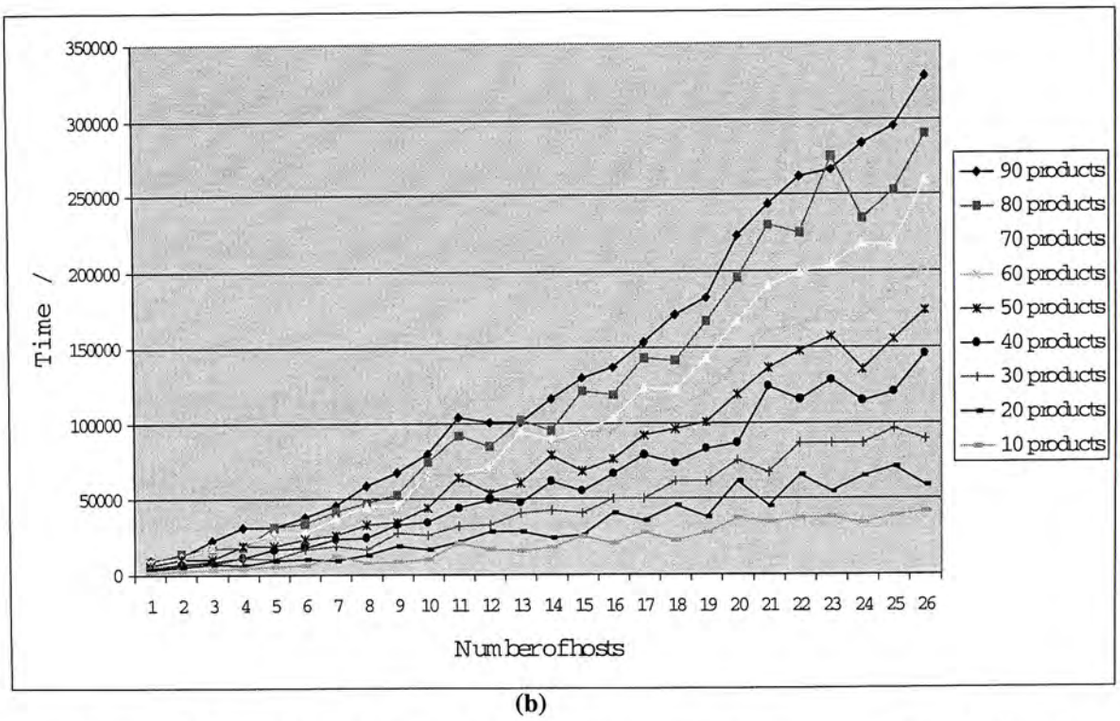
To evaluate the performance of SIAS against the scale of the system, the times required for an agent to travel around an electronic market of different number of hosts, with and without security enforcement, are measured respectively. Queries of different sizes (number of product items) have been tested. The results are plotted in Figures 6.13(a) (without security) and 6.13(b) (with security).

Results show that, the RTT for an agent to travel in SIAS changes more or less linearly over the number of hosts in the system. This is due to the additional time to travel an additional host, and the overhead for each additional host is more or less the same. Moreover, the RTT is also linearly increasing as the number of products of the query increases. This can be explained by the increases in number of database transactions and time to transport an agent.

As an alternative to traditional client/server system, it is interesting to compare the performance of a mobile agent system with an equivalent system deploying client/servers. However, we have not built a client/server equivalent for SIAS, though it is an easy task. This is because we are running SIAS on a network of SUN Sparc machines all connected high speeds. In this case, it is obvious that a simple RPC equivalent can run much faster than SIAS, because the comparatively time-consuming agent interpretation and execution by the



(a)



(b)

Figure 6.13. Round Trip Times of an agent, with different query sizes, against different numbers of hosts in SIAS: (a) without security; (b) with security.

Concordia platform and the underlying Java Virtual Machine can then be skipped. It will be more meaningful to compare the system performances with a client-server equivalent when the communication link between the user and the servers are slow, for example, with a wireless device. In this case, mobile agents (SIAS) can run faster because it can save time

from frequent client-server communication using the slow link. In fact, we are considering this as a direction for future work.

When security is enforced, the RTT increases in general. For the maximum scale of 26 hosts, and maximum size of 90 products in query, the RTT increases by 100 seconds, from 230 seconds to 350 seconds. This can be explained by the extensive use of the RSA algorithm to encrypt and decrypt each item, which is time consuming, especially when the key is long. Therefore, again, we see a trade-off between security and performance in SIAS.

6.6 Reliability Design of SIAS

As stated in the introduction, it was first not our primary interest to study mobile agent reliability, when SIAS was first developed. However, after some times for which the security experiments reported in the previous section was run, it happened that, to our surprise, SIAS fails quite frequently. This made our security experiment could not carry out smoothly, and so we started to tackle the reliability problem of SIAS.

6.6.1 Reliability Problems of SIAS

When we investigated into the problem, we found that the frequent failure of SIAS was actually due to the frequent failure of the Concordia server. When the Concordia server on a particular host fails, there can be two consequences, as described in Chapter 3:

- If the agent we sent is residing on the host with the failing Concordia server, the agent is lost, and any query result carried by the agent will also be lost. Concordia does not provide agent persistence or recovery automatically, so even when the failed server becomes up again, the mobile agent cannot be recovered.
- If the agent we sent is not residing on the host with the failing Concordia server, the agent can keep alive. However, if the agent is going to visit the host with the failed Concordia server, the current host of the agent will still send the agent to the failed

server, without asking for an acknowledgement from the receiving server. Then, the sending server would erase the copy of the agent on the original host, even though a new copy of the agent is not created on the receiving host, due to failure of the Concordia server.

In both cases, the agent does not return to the user, and the SIAS system has failed. This motivated us to build our solutions to make SIAS more fault-tolerant.

6.6.2 Our Solutions to the Problems

We have implemented two measures to make SIAS more fault-tolerant:

- **Forward-echo by agent**

Before an agent is transported to the next destination, the agent sends an "echo" message to the Concordia server on the next host. If there is a reply from the receiving Concordia server, the agent transmission goes on; otherwise, the agent will re-do the echo until the Concordia server replies. This prevents the sending server from mistakenly sending the agent when the receiver is not ready, which makes the agent erased without a new copy created. However, this does not prevent the agent from waiting indefinitely for the receiving server to goes up again.

- **Periodic scan by server**

To prevent an agent from waiting indefinitely due to failure of the receiving server, we implemented a server monitoring system. It is really a shell script running on one of the hosts. The shell script periodically scans each host to see if the Concordia server is running properly on it, similarly to the "echo" by agents. If any host is found with its Concordia server not responding, the script will automatically log into the host, and restart the Concordia server on that host. This ensures that the Concordia server on each host will be at least restarted periodically. Therefore, the chance of indefinite wait of agent for a failed server would be decreased.

These two measures are simple and easy to implement. Moreover, they do not require the agent to change its pre-determined itinerary. There can be other measures to further increase the reliability of SIAS. For example, when an agent finds that the server ahead is not running, it should choose another host to visit; or more complicated ones like replication and logging of agents. There is active research in this area, too. However, for our system, SIAS, as we will see in the next section, these measures already brings a major reliability improvement.

6.6.3 Evaluation of the Reliable SIAS

To evaluate the reliability gain due to the fault-tolerance measures (forward-echo and periodic scan) implemented, we perform an experiment and plot the *reliability curve*, as defined in Section 4.5, of the system. Although it has been found that SIAS fails quite frequently, it may take on average about an hour for 1 out of 20 Concordia servers to be found failed. To exaggerate the failure rates of the system such that the experiment can be carried out faster, we simulate a higher failure rate for the Concordia servers. For each Concordia server, we run a background job every 2 minutes to pick a random integer between 1 and 10. Whenever we get a 10, we terminate the server on that host. Therefore, the failure rate of each Concordia server is about 1/10 per 2 minutes, that is 1/20 per minute. This is really a non-practically high failure rate, and we can see that the saturated failure rate of the system can go up to 100%. However, this does not matter in this experiment, as we are only interested in how much reliability gain can be earned from the measures implemented.

From the results, plotted in Figure 6.14, we see that, neglecting the final bursts, the saturated reliability for the system without reliability implementation is 100%, while that for the system with reliability implementation is decreased to only about 60%. Therefore, there is a 40% increase in reliability.

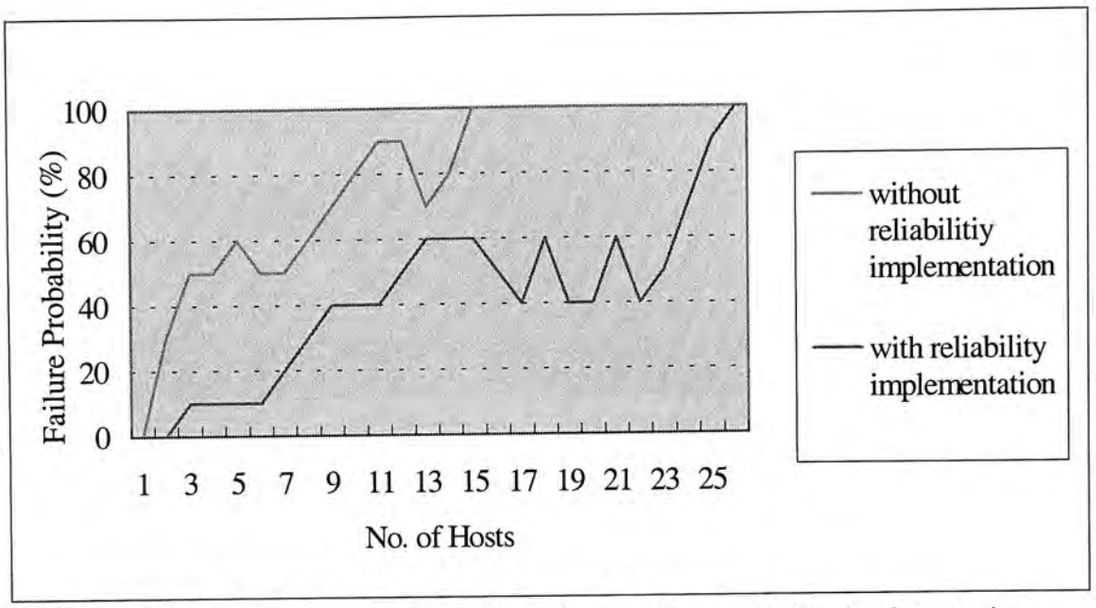


Figure 6.14. Reliability curves of SIAS with and without reliability implementation.

However, the saturated reliability of 60% is still a high value, despite the fact that we have intentionally increased the failure rate of the servers. This can accounted to the not-small possibility that the server hosting the agent has failed. In this case, our measures could not recover the agent, and SIAS will still fail.

Chapter 7

Conclusions and Future Work

This thesis has attempted both theoretical and practical aspects of security and reliability issues of mobile agents. On the theoretical side, a security model and a reliability model were derived for mobile agent systems respectively. On the practical side, an experimental electronic commerce application, SIAS, is developed using mobile agents. The security and reliability problems of the particular application have been studied, and specific solutions to the problems have been devised and implemented.

Modeling is a successful technique for software reliability engineering. It is expected to be more or less directly applicable to mobile agent systems as well. For instance, the simple model for mobile agent reliability developed in this thesis has been applied to analyze the reliability improvement of the fault-tolerance mechanisms implemented for SIAS. However, it is not as simple to apply modeling in the are of security. Effort has been spent on applying the security model to analyze the security mechanisms of SIAS, but it has not been successful. The difficulty is how to capture the attack behavior of malicious hosts.

There has been an opinion that human behavior cannot be modeled, especially true for malicious or irrational behaviors. I agree with this, but I am still optimistic in applying modeling to solve some practical security problems. Although human behaviors are

sometimes unpredictable, the majority of human beings, as a part of society, act according to some social norms. We cannot predict what abnormal behavior might happen, but we can predict how probable abnormal behavior occurs, based on statistics.

For non-critical applications, or not the most critical applications, from a user's point of view, it suffices to make a bet on security for convenience, if there is enough evidence that the risks can be balanced by the expected benefits. Popularity of trust evaluation in current electronic bidding systems solidifies this idea. Users of such systems do not know exactly whether the other party in a transaction is trustworthy or not, but they count their expectation on continuous mutual evaluation from other users. It is interesting to extend this idea to mobile agent systems. The trustworthiness of each host and agent may be obtained by evaluation from other agents or hosts. This is one direction of my future work.

Besides, as we have seen from Chapter 6, it has not been shown, although it is believed, that a mobile agent application can be more efficient than a client/server equivalent. It is interesting to port SIAS to a mobile computing environment, and then build a client/server equivalent to compare the performances of both paradigms. Before the resources are available for this to be done, simulation results should be done to give further evidence for advantages of mobile agents.

Concordia was first chosen as the implementation technology mainly because of its simplicity. This was justified when there was not much knowledge about mobile agents. After more than a year working with Concordia, it turns out that it is neither reliable nor powerful enough for our experiments. Moreover, it does not comply with standards like FIPA and MASIF. To scale up the research on mobile agents, another mobile agent platform should be chosen. One recommendation is Grasshopper, which support both FIPA and MASIF, and is also available in a mobile computing environment, on the Microsoft Windows CE operating system.

In conclusion, mobile agents are going to complement many client/server applications. Enhancing reliability of mobile agent systems surely boost the process, and it is

already taking place actively. On the other hand, mobile agents can be much more useful if the security problem can be tackled. No one can come up with a complete solution for protecting mobile agents, yet no one has proved that this is unsolvable. As more efforts are being spent in this topic, it is believed that the problem can be at least simplified, if not totally solved.

Bibliography

[AS98] The Agent Society. "Agent Code Available for Download". http://www.agent.org/pub/code.html, 1998.

[Bak97] Sean Baker. *CORBA Distributed Objects: Using Orbix.* Addison-Wesley, 1997.

[BGP97] M. Baldi, S. Gai, and G. Picco. "Exploiting code Mobility in Decentralized and Flexible Network Management". In Kurt Rothermel, Radu Popescu-Zeletin, editors, *Mobile Agents, First International Workshop, MA'97, Berlin, Germany, April 1997, Proceedings, LNCS 1219*, p. 13-26. Springer, 1997.

[Bir96] Kenneth P. Birman. *Building Secure and Reliable Network Applications.* Manning, 1996.

[BLOJ94] Sarah Brocklehurst, Bev Littlewood, Tomas Olovsson and Erland Jonsson. "On Measurement of Operational Security". In *Proceedings of the Ninth Conference on Computer Assurance (COMPASS'94): Safety, Reliability, Fault Tolerance and Real Time, Security*, p.257-266.

[BP98] L. Bernado, P. Pinto. "Scalable Service Deployment Using Mobile Agents". In Kurt Rothermel, Fritz Hohl, editors, *Mobile Agents, Second International Workshop, MA'98, Stuttgart, Germany, September 1998, Proceedings, LNCS 1477*, p. 261-272. Springer, 1998.

[CL99] Anthony H. W. Chan and Michael R. Lyu. "Security Modeling and Evaluation for the Mobile Code Paradigm". In P. S. Thiagarajan, R. Yap (Eds.): *Advances in Computing Science - ASIAN'99 Proceedings of 5th Asian Computing Science Conference*, Phuket, Thailand, December 1999, pp. 371-372, Springer-Verlag (LNCS 1742), 1999.

[CL00] Anthony H. W. Chan and Michael R. Lyu. "The Mobile Code Paradigm and Its Security Issues". In Gilbert H. Young (Ed.): *World Wide Web: Technologies and Applications for the New Millennium*, p.353-357, CSREA Press, 2000.

[CWWL00a] Anthony H. W. Chan, T. Y. Wong, Caris K. M. Wong, and Michael R. Lyu. "SIAS: A Secure Shopping Information Agent System". In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein (Eds.): *Agents 2000 - Proceedings of the Fourth International Conference on Autonomous Agents*, p. 257-258, ACM Press, 2000.

[CWWL00b] Anthony H. W. Chan, Caris K. M. Wong, T. Y. Wong, and Michael R. Lyu. "Design, Implementation, and Experimentation on Mobile Agent Security for Electronic Commerce Applications". To be published in the 2000 International Conference on Parallel and Distributed Processing Techniques

and Applications (PDPTA'2000), Monte Carlo Resort, Las Vegas, Nevada, USA, June 26-29, 2000.

[CWWL00c] Anthony H. W. Chan, Caris K. M. Wong, T. Y. Wong, and Michael R. Lyu. "Securing Mobile Agents in Electronic Commerce: an Experiment". To be published in the 15th Internatioanl Conference on Information Security (SEC2000) of the World Computer Congress 2000 (WCC2000), Beijing, China, August 21-25, 2000.

[FKK96] Alan O. Freier, Philip Karlton, and Paul C. Kocher, "The SSL Protocol Version 3.0". Internet Draft. http://home.netscape.com/eng/ssl3/draft302.txt, November 18, 1996.

[GBH98] Michael S. Greenberg, Jennifer C. Byington, and David G. Harper. "Mobile Agents and Security". In *Volume 367, IEEE Communications Magazine.* IEEE Press, July 1998.

[GLT98a] S.S. Gokhale, M.R. Lyu, and K.S. Trivedi, "Reliability Simulation of Component-Based Software Systems". In *Proceedings of Ninth International Symposium on Software Reliability Engineering (ISSRE'98), Paderborn, Germany, November 4-7 1998*, p. 192-201.

[GLT98b] S.S. Gokhale, M.R. Lyu, and K.S. Trivedi. "Software Reliability Analysis Incorporating Fault Detection and Debugging Activities". In *Proceedings of Ninth International Symposium on Software Reliability Engineering (ISSRE'98), Paderborn, Germany, November 4-7 1998*, p. 202-211.

[GO79] A.L. Goel, and K. Okumoto. "Time-Dependent Error-Detection Rate Model for Software and Other Performance Measures". In *IEEE Transactions on Reliability, vol. R-28, no.3, August 1979*, p.206-211.

[GM98] General Magic Incorporation. "Agent Technology". http://www.genmagic.com/technology/mobile_agent.html, 1998.

[GV97] Carlo Ghezzi and Giovanni Vigna. "Mobile Code Paradigms and Technologies: A Case Study". In Kurt Rothermet, Radu Popescu-Zeletin, editors, *Mobile Agents, First International Workshop, MA '97, Berlin, Germany, April 1997, Proceedings, LNCS 1219*, p. 39-49. Springer, 1997.

[Hoh98a] Fritz Hohl. "Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts". In Giovanni Vigna, editor, *Mobile Agents and Security, LNCS 1419*, p. 92-113. Springer, 1998.

[Hoh98b] Fritz Hohl. "A Model of Attacks of Malicious Hosts Against Mobile Agents". In *Fourth Workshop on Mobile Object Systems (MOS'98): Secure Internet Mobile Computations*, http://cuiwww.unige.ch/~ecoopws/ws98/papers/hohl.ps, 1998.

[Hoh98c] Fritz Hohl. "Mobile Agent Security and Reliability". In *Proceedings of the Ninth International Symposium On Software Reliability Engineering (ISSRE'98), page 181.* IEEE Computer Society, 1998.

[IBM99] "IBM Aglets Software Development Kit Homepage". http://www.trl.ibm.co.jp/aglets/

[Inp98] Inprise Inc.. Visibroker, CORBA Technology from Inprise. http://www.borland.com/visibroker/, 1998

[IPV98] IPVR, University of Stuttgart. 'The Home of Mole". http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html, 1999.

[Jon97] Erland Jonsson. "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior". In *IEEE Transactions on Software Engineering, Vol. 23, No. 4.* IEEE, April 1997.

[Jon99] Erland Jonsson. "An Integrated Framework for Security and Dependability". http://www.ce.chalmers.se/staff/jonsson/paradigms_nspw98.fm55.pdf, 1999.

[KLO97] Günter Karjoth, Danny B. Lange, and Mitsuru Oshima. "A Security Model for Aglets". In Giovanni Vigna, editor, *Mobile Agents and Security, LNCS 1419*, p. 188-205. Springer, 1998.

[KT99] Neeran M. Karnik and Anand R. Tripathi. "Security in the Ajanta Mobile Agent System". Technical Report, Department of Computer Science, University of Minnesota, May 1999.

[Lin97] J. Linn. "Generic Security Service Application Program Interface, Version 2". RFC2078 (Obsoletes: RFC1508), 1997.

[LO99] Danny b. Lange and Mitsuru Oshima. "Seven Good Reasons for Mobile Agents". In *Communications of the ACM*, p.88-89, 1999 March.

[MBB+98] Dejan Milojicic, Markus Breugst, Ingo Busse, John Campbell, Stefan Covaci, Barry Friedman, Kazuya Kosaka, Danny Lange, Kouichi Ono, Mitsuru Oshima, Cynthia Tham, Sankar Virdhagriswaran, and Jim White. "MASIF: The OMG Mobile Agent System Interooperability Facility". In Kurt Rothermel, Fritz Hohl, editors, *Mobile Agents, Second International Workshop, MA'98, Stuggart, Germany, September 1998, Proceedings, LNCS 1477*, p. 50-67. Springer, 1998.

[Mit99a] Mitsubishi Electronics ITA. "Concordia - Java Mobile Agent Technology". http://www.meitca.com/HSL/Projects/Concordia/

[MIT99b] Massachusetts Institute of Technology. " Kerberos: The Network Authentication Protocol". http://web.mit.edu/kerberos/www/

[ODK97] R. Ortalo, Y. Deswarte and M. Kaaniche, "Experimenting with Quantitative Evaluation Tools for Montoring Operational Security". In *Proceedings of DCCA-6*, Grainau (D), March 5-7, IEEE CS Press, *Series on Dependable Computing and Fault Tolerant Systems, vol.1*1, pp. 307-328.

[Ohb84] M. Ohba. " Software Reliability Analysis Models". In *IBM Journal of Research and Development, vol. 21, no. 4, July 1984*, p. 428-443.

[OMG98] Object Management Group. "CORBA Security Service Specification (1.2)". http://www.omg.org/corba/sectrans.html#sec, December 1998.

[OOC97] L. A. G. Oliveira, P. C. Oliveira, and E. Cardozo. "An Agent-based Approach for Quality of Service Netgotiation and Management in Distributed Multimedia Systems". In Kurt Rothermet, Radu Popescu-Zeletin, editors, *Mobile Agents, First International Workshop, MA'97, Berlin, Germany, April 1997, Proceedings, LNCS 1219*, p. 1-12. Springer, 1997.

[PK98] Vu Anh Pham and Ahmed Karmouch, "Mobile Software Agents: An Overview". In *Volume 367, IEEE Communications Magazine*. IEEE Press, July 1998.

[RN95] Stuart Russell and Peter Norvig. "Intelligent Agents". In *Artificial Intelligence, A Modern Approach*, p.31-50. Prentice Hall, 1995.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, 1978 Feb.

[Sch96] Bruce Schneier. *Applied Cryptography*. Wiley, 2nd edition, 1996.

[Som92] Ian Sommerville. *Software Engineering*, 4th edition. Addison-Wesley, 1992.

[ST98] Tomas Sander and Christian F. Tschudin. "Protecting Mobile Agents Against Malicious Hosts". In Giovanni Vigna, editor, *Mobile Agents and Security, LNCS 1419*, p. 44-60. Springer, 1998.

[Sta99] William Stallings. *Cryptography and Network Security, Principles and Practice*. Prentice Hall, 2nd edition, 1999.

[Sun99] Sun Microsystems. "Java Security Architecture". http://java.sun.com/products//jdk/1.2/docs/guide/security/spec/security-specTOC.fm.html

[Tsc99] C. Tschudin. "Mobile Agent Security", *Intelligent Information Agents: Agent Based Information Discovery and Management in the Internet*, p. 431 - 446, Springer, 1999.

[Whi98] Jim White. "Mobile Agents White Paper". http://www.genmagic.com/technology/techwhitepaper.html, 1998.

[WPW98] Walsh, Tom; Paciorek, Noemi; Wong, David. "Security and Reliability in Concordia". In *Proceedings of the 31st Annual Hawaii International Conference on System Sciences (HICSS31)*, 1998.

[YOO83] S. Yamada, M. Ohba, and S. Osaki. "S-Shaped Reliability Growth Modelling for Software Error Detection". In *IEEE Transactions on Reliability, vol. R-32, no. 5, December 1983*, p.475-478.